Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

(12)

REPORT DOCUMENTATION PAGE

AD-A230 380

DTIC ELECTE DEC 28 1990 S C S

LE COPY

1a. REPORT SECURITY CLASSIFICATION Unclassified	
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR 492	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-89-J-1988
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD	8b. OFFICE SYMBOL (If applicable)
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Results in Computational Geometry: Geometric Embeddings and Query-Retrieval Problems.

12. PERSONAL AUTHOR(S)

Mark David Hansen

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	November 1990	95

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Many fundamental questions in computational geometry arise from the consideration of distributions of points in euclidean space. This thesis explores two important areas of computational geometry in this setting: geometric embeddings and query-retrieval problems. Each area is addressed in a separate part of the thesis.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Carol Nicolora	22b. TELEPHONE (Include Area Code) (617) 253-5894 / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆U.S. Government Printing Office: 1985—507-047

Unclassified

Part I examines the geometric embedding problem for many of the graphs which are important in the study of parallel computation [33]. Given an undirected graph G with n vertices, and a set P of n points in the plane, the geometric embedding problem consists of finding a bijection from the vertices of G to the points in the plane which minimizes the sum total of edge lengths of the embedded graph. We give fast approximation algorithms for embedding d-dimensional grids in the plane which are within a factor of $O(\log n)$ times optimal cost for $d > 2$ and $O(\log^2 n)$ for $d = 2$. We also show that any embedding of a hypercube, butterfly, or shuffle-exchange graph must be within an $O(\log n)$ factor of optimal cost. When the points of P are randomly distributed, or arranged in a grid, we give a polynomial time algorithm which can embed arbitrary weighted graphs in these points with cost within an $O(\log^2 n)$ factor of optimal. Many of these results extend to higher dimensions when $P \subset R^d$.

Aside from the intrinsic mathematical interest of these problems, they also have applications in the field of parallel processing. For example, we show how the algorithms which we develop for geometric embeddings can be used to give solutions which are within an $O(\log^2 N)$ factor of optimal to problems of performance optimization for array-based parallel processors in the following areas: communication load balancing, dynamic allocation of jobs to processors, reconfiguring around faults, and simulating other architectures.

Part II of this thesis examines query-retrieval problems concerning distributions of points in euclidean space. In this part, we describe a new technique for solving a variety of query-retrieval problems in optimal time with optimal or near-optimal space [2]. In particular, we use the technique to construct algorithms and data structures for circular range searching, half-space range searching, and computing k-nearest neighbors in a variety of metrics. For each problem and each query, the response to the query is provided in $O(k)$ or $O(k + \log n)$ time where k is the size of the response and n is the size of the problem. (E.g., for the n-point k-nearest neighbors problem, the k-nearest neighbors of any query point are provided in $O(k + \log n)$ steps.) Depending on the problem being solved, the space required for the data structure is either linear or $O(n \log n)$. Hence, the time bounds are optimal and the space bounds are optimal or near-optimal. Previously known data structures for these problems required a factor of $\Omega(\log n(\log\log n)^2)$ or $\Omega(\log n \log\log n)$ more space and/or more time to answer each query.

Our compaction technique incorporates planar separators, filtering search, and the probabilistic method for discrepancy problems. The fundamental idea is that k^{th}-order Voronoi diagrams (and other suitable proximity diagrams) can be compacted from $k^{O(1)}n$ space to $O(n)$ space and still retain all the information that is essential for solving query problems.

Results in Computational Geometry: Geometric Embeddings and Query-Retrieval Problems

by

Mark David Hansen

A.B., Mathematics
Cornell University
(1986)

M.S., Mathematics
University of Chicago
(1987)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1991

Signature of Author .....
Department of Mathematics
January 11, 1991

Certified by .....
F. Thomson Leighton
Professor of Applied Mathematics
Thesis Supervisor

Accepted by .....
Daniel J. Kleitman, Chairman
Applied Mathematics Committee

Accepted by .....
Sigurdur Helgason, Chairman
Departmental Graduate Committee

Results in Computational Geometry: Geometric Embeddings and Query-Retrieval Problems

by

Mark David Hansen

Submitted to the Department of Mathematics
on January 11, 1991, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Many fundamental questions in computational geometry arise from the consideration of distributions of points in euclidean space. This thesis explores two important areas of computational geometry in this setting: geometric embeddings and query-retrieval problems. Each area is addressed in a separate part of the thesis.

Part I examines the geometric embedding problem for many of the graphs which are important in the study of parallel computation [33]. Given an undirected graph G with n vertices, and a set P of n points in the plane, the geometric embedding problem consists of finding a bijection from the vertices of G to the points in the plane which minimizes the sum total of edge lengths of the embedded graph. We give fast approximation algorithms for embedding d-dimensional grids in the plane which are within a factor of $O(\log n)$ times optimal cost for $d > 2$ and $O(\log^2 n)$ for $d = 2$. We also show that any embedding of a hypercube, butterfly, or shuffle-exchange graph must be within an $O(\log n)$ factor of optimal cost. When the points of P are randomly distributed, or arranged in a grid, we give a polynomial time algorithm which can embed arbitrary weighted graphs in these points with cost within an $O(\log^2 n)$ factor of optimal. Many of these results extend to higher dimensions when $P \subset R^d$.

Aside from the intrinsic mathematical interest of these problems, they also have applications in the field of parallel processing. For example, we show how the algorithms which we develop for geometric embeddings can be used to give solutions which are within an $O(\log^2 N)$ factor of optimal to problems of performance optimization for array-based parallel processors in the following areas: communication load balancing, dynamic allocation of jobs to processors, reconfiguring around faults, and simulating other architectures.

Part II of this thesis examines query-retrieval problems concerning distributions of points in euclidean space. In this part, we describe a new technique for solving a variety of query-retrieval problems in optimal time with optimal or near-optimal space [2]. In particular, we use the technique to construct algorithms and data structures for circular range searching, half-space range searching, and computing k-nearest neighbors in a variety of metrics. For each problem and each query, the response to the query is provided in $O(k)$ or $O(k + \log n)$ time where k is the size of the response and n is the size of the problem. (E.g., for the n-point k-nearest neighbors problem, the k-nearest neighbors of any query point are provided in $O(k + \log n)$ steps.) Depending on the problem being solved, the space required for the data structure is either linear or $O(n \log n)$. Hence, the time bounds are optimal and the space bounds are optimal or near-optimal. Previously known data structures for these problems required a factor of $\Omega(\log n(\log \log n)^2)$ or $\Omega(\log n \log \log n)$ more space and/or more time to answer each query.

Our compaction technique incorporates planar separators, filtering search, and the probabilistic method for discrepancy problems. The fundamental idea is that k^{th}-order Voronoi diagrams (and other suitable proximity diagrams) can be compacted from $k^{O(1)}n$ space to $O(n)$ space and still retain all the information that is essential for solving query problems.

Thesis Supervisor: F. Thomson Leighton
Title: Professor of Applied Mathematics

Contents

Acknowledgments

A thesis is as much a product of a student's environment as of his own mind. Given this, I owe a great deal of thanks to the theoretical computer science community at MIT. The caliber of scholarship and camaraderie which I have known here has been exceptional.

Among all the talented individuals within this community, my advisor Tom Leighton stands out in many respects. His quick mind and deep mathematical insight have proven invaluable for suggesting areas of research and innovative approaches to problem solving. In addition, I am grateful for Tom's support of all my academic endeavors, both in mathematics and elsewhere at MIT.

I also owe a great deal of thanks to Alok Aggarwal. As an advisor and friend, he has been an enthusiastic supporter. Alok introduced me to the field of computational geometry and much of the work in this thesis was done jointly with him.

Finally, I would like to thank Mike Sipser for reading the initial draft of this thesis and serving on my committee.

During the past three years, I have found that most of the learning which occurs at MIT happens not in the classroom, but from conversations and joint work with fellow graduate students. In addition to being great hockey and softball teammates, the following people contributed significantly to my understanding of computer science and to the research contained in this thesis: Avrim Blum, Tom Cormen, Bruce Maggs, Seth Malitz, James Park, Satish Rao, Eric Schwabe, and Cliff Stein.

Anybody who has spent any time on the third floor of LCS knows that the community could not function without Be Hubbard. I've really appreciated her help and

sense of humor over the past three years. I am also very grateful to Phyllis Ruby for guiding me through the bureaucratic channels of MIT and helping to keep me in good standing with the math department administration.

In terms of financial support, I would like to thank the National Science Foundation for a three year graduate fellowship and the DARPA Research Program in Parallel Processing for a research assistantship during my final year at MIT.

These acknowledgments could not be complete without thanking Lorraine Chow for the years of love and support she has given me during my graduate school career. These years would not have been nearly as happy if I had not had her to share them with me.

Finally, I would like to thank my family for making this thesis possible. When my father, David, passed away eight years ago, I was just beginning college as a Cornell freshman. Three years later, when I began to plan for graduate school, I would often remember the following incident from my childhood: As a boy, I used to enjoy helping my father work in his woodshop. Unfortunately, I never had the discipline to learn to use his tools properly and as a result my woodworking projects were rarely very successful. One day, while I was trying to build a simple birdhouse, I mentioned that when I grew up I hoped to have a Ph.D. like he did. Looking at my shoddy birdhouse, with bent nails sticking out all over it, he said, "Mark, I'm worried that you won't have the patience and discipline you need to get through graduate school."

I have to admit that my father's remark was pretty close to the truth and were he here today, I'm sure this thesis would surprise him. What he underestimated, though, was not my own self-discipline, but the love, strength, and support which I would receive from the family he and my mother had raised. Time and again over the past four years, my mother Ann, brother Chris, and sister Leigh, have been there for me when I needed them. Since I can never thank them enough for all they have done, I only hope that this work reflects the high standards of scholarship and character which my family has taught me.

For my parents, Ann and David Hansen.

Introduction

This thesis examines problems in computational geometry which arise from the consideration of distributions of points in euclidean space. The fundamental case concerns a set of n points $P = p_1, p_2, \ldots, p_n$ in the euclidean plane. The class of problems we present involves constructing the optimal embedding of of a given graph, such as a mesh, in P. Such problems arise, for example, when one tries to determine how to optimally link together a group of distributed computers into a mesh or hypercube network. This class of problems is referred to as the *geometric embedding problem.* In general, geometric embedding problems are NP-complete. In this thesis, we give approximation algorithms which yield nearly optimal solutions to a large variety of geometric embedding problems in polynomial time.

The second class of geometry problems we consider are *query-retrieval problems.* In this case, we are interested in "retrieving" the answers to a set of queries about P. For example: "Among the n points in the plane, find the k points that are closest to position (x_i, y_i) for $i = 1 \ldots m$." The task is to preprocess P and devise a data structure so that each query can be answered as quickly as possible. In this thesis, we are primarily concerned with minimizing the space used by the query-retrieval data structure, while maintaining optimal time answers to individual queries.

Geometric Embeddings

Part I of this thesis describes the geometric embedding results. In Chapter 1 a precise definition of the geometric embedding problem is given, the major contributions of this thesis in that area are described, and applications of this work to solving parallel processing problems are discussed. Chapters 2 and 3 describe approximation algorithms for near optimal embeddings of graphs important to the theory of parallel computation: the hypercube, butterfly, shuffle-exchange graph, and mesh. Chapter 4 presents another class of approximation algorithms for geometric embeddings which are based on a powerful separator theorem due to Leighton and Rao [38]. These algorithms give near-optimal embeddings of *any* graph in the plane, provided some assumptions are made about the distribution of points P. Finally, in Chapter 5 we describe in detail the relationship between the embedding results and problems in parallel computation.

Query-Retrieval Problems

Part II of this thesis describes the query-retrieval results. In Chapter 6 we give some background on the importance of data structures in the solution of query-retrieval problems. The main applications to k-nearest neighbor search, circular range search, and half-space range search are then described. Chapter 7 contains a description of the fundamental compaction technique which allows us to greatly reduce the space required to store a Voronoi diagram's essential information for solving query-retrieval problems. Lastly, in Chapter 8, the application of this compaction technique to the three problems described in Chapter 6 is presented in detail.

Part I

Geometric Embeddings

Chapter 1

Overview of Geometric Embedding Results

1.1 Geometric Embeddings

A *geometric embedding* of $G = (V, E)$ into a set of points P in euclidean space is a bijection $f : V \rightarrow P$. We are interested in finding f which minimizes the total edge length of the graph induced on P: $\sum_{(u,v)\in E} dist(f(u), f(v))$. Here, *dist* is the euclidean metric. The most famous example of a geometric embedding problem is the *euclidean traveling salesman problem.* In this case, G is a cycle on N nodes and many approximation algorithms for finding the near-optimal cost embedding of G into points in R^2 are known. One example of such an approximation algorithm is due to Christofides [23]. In this paper, we will consider the geometric embedding problem for graphs which are important in the theory of parallel computation: grids, hypercubes, butterflies, and shuffle exchange graphs.

We will also consider some special cases of the geometric embedding problem in the plane and higher dimensional spaces when the set of points P are distributed randomly, or arranged in a grid. With these restrictions on P we are able to address the more general problem of embedding weighted, undirected graphs in R^d. For a

weighted graph, our goal is to minimize the weighted sum of edge lengths in the graph induced on P: $\sum_{(u,v)\in E} w(u,v)dist(f(u),f(v))$. Here, $w(u,v)$ is the weight of edge $(u,v) \in E$. In this case we may also consider *dist* to be the manhattan (L_1) metric. Some recent results in multicommodity flow [38] allow us to achieve near-optimal embeddings of arbitrary weighted graphs in these two special cases. All of these results have important applications to parallel processing.

1.2 Mathematical Results

Polynomial time algorithms for finding optimal or $O(\log N)$ times optimal embeddings in R^d of simple structures such as cycles, trees, or stars are familiar to many researchers [11, 23]. In this paper, we study the problem of embedding more complicated graphs in euclidean space. The structures we study include many of the important graphs from the theory of parallel computation: hypercubes, butterflies, shuffle-exchange graphs, meshes, cubes, and higher dimensional grids.

A straightforward application of known routing results proves that any embedding of the high-bandwidth, low diameter hypercube-like graphs is within an $O(\log N)$ factor of optimal. Meshes, cubes, and higher-dimensional grids do not have the nice routing properties of these graphs and therefore embedding them efficiently constitutes a significantly more difficult problem. Section 3 presents the analytic techniques which enable us to give a fast algorithm for embedding these graphs within an $O(\log N)$ or $O(\log^2 N)$ factor of optimal. Some of these techniques can be generalized and used together with separator approximation algorithms [38] to give near-optimal embeddings of arbitrary weighted graphs into uniform distributions of points in euclidean space.

The embedding results in this paper are grouped into two categories based on the arrangement of the set of points P into which we are embedding the graph G. P could be (1) an arbitrary set of points, or (2) a array of points or a random set of

uniformly distributed points.

1.2.1 Arbitrary Points

For hypercubes, butterflies, and shuffle-exchange graphs it is straightforward to show that any embedding is near-optimal. In fact, in this case, we need not even assume that the points P are in the plane, but only that they are in some graph which obeys the triangle inequality.

Theorem 1.2.1 *Given a graph G which is an N-node hypercube, butterfly, or shuffle-exchange graph, and a set P of points connected by edges which obey the triangle inequality, any embedding of G into P has cost within an $O(\log N)$ factor of optimal.*

For grids, cubes, and higher d-dimensional grids, the problem is substantially more difficult, and we give a fast algorithm for near optimal embeddings in the plane.

Theorem 1.2.2 *There exists an algorithm which, when given a d-dimensional grid G on N nodes and a set P of N points in the plane, in $O(N \log N)$ time finds a geometric embedding with cost a factor of $O(\log^2 N)$ times optimal when G is a square mesh ($d = 2$) and $O(\log N)$ times optimal when G is a cube or higher dimensional grid ($d > 2$).*

In fact, Theorem 1.2.2 generalizes to the case $P \subset R^k$ in the following manner. We can embed grids of dimension k into R^k within an $O(\log^2 N)$ factor of optimal cost. and grids of dimension $d > k$ within an $O(\log N)$ factor.

1.2.2 Arrays of Points and Randomly Distributed Sets of Points

We can achieve significantly more general embedding results when we place some restrictions on the arrangement of the points P. If P is a set of evenly spaced points

arranged in a d-dimensional array in R^d for some fixed constant d, then we can use the graph separator approximation results of Leighton and Rao [38] to construct a polynomial time algorithm which produces a geometric embedding of an arbitrary weighted graph G into P with cost within an $O(\log^2 N)$ factor of optimal.

Theorem 1.2.3 *There exists a polynomial time algorithm which, when given an arbitrary N-node weighted graph G and a d-dimensional array of N points P in R^d, embeds G in P with cost within an $O(\log^2 N)$ factor of optimal.*

The same theorem holds, with high probability, when P is a random set of uniformly distributed points in R^d.

Theorem 1.2.4 *There exists a polynomial time algorithm which, when given an arbitrary N-node graph G and a random set of N uniformly distributed points P in R^d, with high probability embeds G in P with cost within an $O(\log^2 N)$ factor of optimal.*

1.3 Applications

1.3.1 Efficient Parallel Computation

Efficient parallel processing requires effective algorithms for embedding large computational problems into parallel architectures. Given a *computation graph* for a parallel algorithm which consists of process nodes and edges between nodes whose processes exchange data, and a fixed parallel machine architecture, we would like to embed this computation graph into the architecture in a manner which makes efficient use of the machine's interconnect network for inter-process communication. A poor embedding could result in an assignment of process nodes to processors which clogs the network with inter-process communication and greatly degrades running time.

We measure the efficiency of an embedding of a computation graph into a parallel architecture in terms of the *communication load* this embedding induces on the ma-

chine. Communication load is a measure of the total volume of traffic an algorithm produces on a network, and is defined to be the sum total of distances that interprocess messages travel in the interconnect network. As a corollary of Theorem 1.2.3 we show how to embed *any* computation graph in an array-based parallel machine with communication load within an $O(\log^2 N)$ factor of optimal. This corollary can also be interpreted to mean that we can reconfigure an array-based machine to simulate any other architecture within an $O(\log^2 N)$ factor of optimal communication load.

Theorems 1.2.1, 1.2.2, and 1.2.4 allow us to give algorithms for the efficient dynamic allocation of resources on a multiprocessor. Suppose that our machine has a number of users who are continually submitting jobs for parallel processing. The machine's operating system dynamically allocates processors to jobs, and as jobs finish, *holes* of idle processors open up in the network. For arbitrary holes in an array-based processor, our results give algorithms for embedding jobs with near-optimal communication load when these jobs have common computation graphs such as grids, cubes, hypercubes, shuffle-exchange graphs, or butterflies. When the holes are assumed to be randomly and uniformly distributed, we can give efficient embedding algorithms for any computation graph.

1.3.2 Network Reconfiguration and Wafer-Scale Integration

Consider a nation-wide network, similar to one operated by the weather service, of computation centers linked over standard telecommunication channels. If the network is tracking a storm system, it will need to be continually reconfigured as centers enter and drop out. Theorems 1.2.1, 1.2.2, and 1.2.4 provide algorithms for dynamically reconfiguring in a manner which produces networks with near-optimal communication overhead.

In *wafer-scale integration* [37] reconfiguring the live cells on a silicon wafer into

a usable network comprises a geometric embedding problem. Similarly, consider reconfiguring the live processors on an array-based machine with faults. Our results indicate how to reconfigure for near-optimal communication load on the wafer or array in the presence of arbitrary faults when the embedded network is a grid, cube, or hypercube-like graph. If faults are random or uniformly distributed, we can reconfigure well for any embedded network.

1.4 Related Work

Several special cases of finding optimal solutions to the geometric embedding problem have been proven NP-hard. These include the cases when G is a traveling salesman tour or binary tree [32]. Christofides' approximation algorithm for the geometric traveling salesman achieves embeddings which are 1.5 times optimal. Papadimitriou and Yanakakis [40] have further shown that the geometric embedding problem is NP-hard for large classes of trees.

In a recent paper, Bern, Karloff, Raghavan, and Schieber [11] studied geometric embeddings of binary trees, cycles, and stars. They present a number of interesting results including very fast approximation algorithms for embedding these graphs in the line and plane. The focus of their paper is on finding fast algorithms for efficiently embedding these simple structures. In contrast, the present paper concentrates on finding near-optimal embedding techniques for some of the more complex graphs studied in the theory of parallel processing. In the case of d-dimensional grids, the approximation algorithm we present also happens to be very fast.

1.5 Outline of Part I

The remainder of Part I is divided into four chapters. Chapter 2 presents Theorem 1.2.1 as a straightforward corollary of some well-known routing results. Here we indicate the connection between routing results and geometric embeddings which

will be used in the analysis of the grid embedding algorithm. Chapter 3 presents the approximation algorithm and mathematical analysis necessary to prove Theorem 1.2.2. Chapter 4 contains the probabilistic analysis and approximation algorithms for Theorems 1.2.3 and 1.2.4. Finally, Chapter 5 discusses some applications of these theorems to parallel processing.

Chapter 2

Hypercube Related Graphs

In this chapter we use well known routing results to prove Theorem 1.2.1 and show that any embedding of the high-bandwidth, low-diameter hypercube-like graphs is within an $O(\log N)$ factor of optimal. Notice that the proof given below does not use any properties of euclidean space, but only requires that the metric being embedded into obey the triangle equality.

Theorem 1.2.1 Given a graph G which is an N-node hypercube, butterfly, or shuffle-exchange graph, and a set P of points connected by edges which obey the triangle inequality, any embedding of G into P has cost within an $O(\log N)$ factor of optimal.

Proof: Let P be a complete graph on N points with edge weights which obey the triangle inequality. (P being points in the plane with the euclidean metric is hence a special case.) For any subset E of edges in P, let $\|E\|$ be the sum of the weights of these edges.

First consider the N-node hypercube H. Let $f_{opt} : H \rightarrow P$ be an optimal embedding of a hypercube in P. Then let $g : H \rightarrow P$ be any other embedding. Define $f_{opt}(H)$ and $g(H)$ to be the hypercube subgraphs of P induced by the edges of H. Group the edges of H by dimension, so that we have groups E_1, ..., $E_{\lg N}$ where the

edges in E_i connect points across the i^{th} dimension. Then $g(E_i)$ defines a matching in P. The following well-known lemma due to Benes [9] indicates that we can connect up the pairs in this matching via paths in $f_{opt}(H)$, using each edge of $f_{opt}(H)$ at most $O(1)$ times:

Lemma 2.0.1 (Benes) *Any permutation of N nodes in the N-node hypercube H can be routed so that each edge of H is used at most $O(1)$ times.*

Hence, by the triangle inequality, $\|g(E_i)\| \leq O(\|f_{opt}(H)\|)$. Summing over the $\log N$ dimensions, we see that $\|g(H)\| \leq O(\lg N)\|f_{opt}(H)\|$.

Now let B be the N-node butterfly. Again, let $f_{opt} : B \rightarrow P$ be an optimal embedding of a butterfly in P and let $g : B \rightarrow P$ be any other embedding. Then the edges of $g(B)$ can be decomposed into 4 disjoint partial matchings. (B has degree 4). Another familiar routing lemma [9] shows that each matching can be routed in $f_{opt}(B)$ using edges at most $O(\log N)$ times:

Lemma 2.0.2 (Benes) *Any permutation of N nodes in the N-node butterfly B, or N-node shuffle exchange graph S, can be routed so that each edge of B or S is used at most $O(\log N)$ times.*

Hence, using the triangle inequality $\|g(B)\| = O(\log N)\|f_{opt}(B)\|$. The proof for the shuffle exchange graph is similar. □

Grids do not have the high bandwidth and low diameter of the three graphs studied above. Hence we cannot expect to get an $O(\log N)$ factor approximation algorithm for embedding grids using these simple routing ideas alone. We will, however, use routing results in the analysis of our approximation algorithm for grids.

Chapter 3

d-Dimensional Grids

3.1 Overview

Chapter 3 is devoted to the proof of Theorem 1.2.2.

Theorem 1.2.2 There exists an algorithm which, when given a d-dimensional grid G on N nodes and a set P of N points in the plane, in $O(N \log N)$ time finds a geometric embedding with cost a factor of $O(\log^2 N)$ times optimal when G is a square mesh ($d = 2$) and $O(\log N)$ times optimal when G is a cube or higher dimensional grid ($d > 2$).

We begin by using some routing results on grids to prove that certain sub-regions of the plane are always densely packed with points. A divide and conquer subroutine is then invoked to embed pieces of the grid into these dense regions. This subroutine is proven to give nearly optimal embeddings of these pieces provided the image points are contained in a sufficiently dense region. Finally, we show that these pieces can be hooked together to form an embedding of the entire grid with near-optimal cost.

3.2 A Review of Routing Results for Grids

Within constant factors, the best routing result which can be achieved for d-dimensional grids states that:

Lemma 3.2.1 *Any permutation of an m point subset of a d-dimensional grid G can be routed using each edge of G at most $2\lceil m^{\frac{1}{d}} \rceil$ times.*

In order to prove Lemma 3.2.1, we begin by proving a somewhat less general result concerning the congestion required to route permutations of an entire d-dimensional grid:

Lemma 3.2.2 *Any permutation of the $N = n^d$ points of a d dimensional grid G can be routed using each edge at most $2N^{\frac{1}{d}}$ times.*

Proof: The proof proceeds by induction on d. For $d = 1$ we have a line, and any greedy routing will work. Now suppose that the theorem holds for $d - 1$. Pick a dimension and consider G to be composed of n $d - 1$ dimensional planes connected by columns along that dimension. Each point is contained in a unique column with $n - 1$ other points. Consider each source and destination pair of the permutation. By induction we will be finished if we can permute points within their columns so that each source and destination pair end up in the same plane. For at this stage we would then have n similar routing problems on the n planes. Each edge of the columns in our chosen dimension would have been used at most $2n = 2N^{\frac{1}{d}}$ times, and later stages in the routing never use that dimension again. So it suffices to show that we can permute the points within a column so source and destination points always end up in the same plane.

Consider a bipartite graph with one node on each side for each column. Draw an edge for each source and destination pair between any two columns. See Figure 3-1.

The result is a bipartite graph on 2n nodes which is n-regular. It is well known that such a graph can be n-colored. That is, color the edges of the graph with n

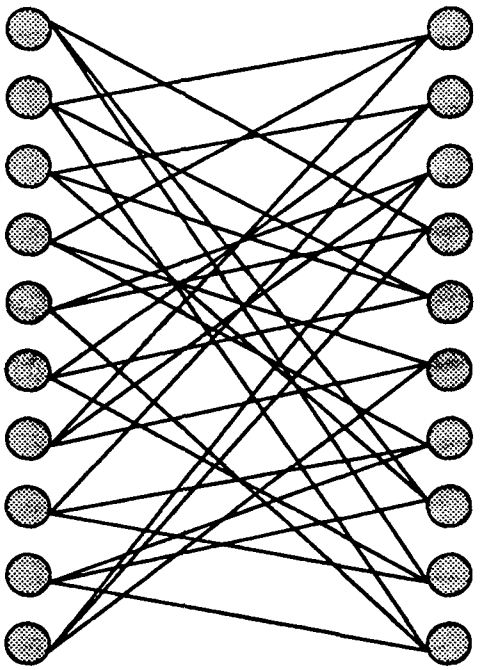

Figure 3-1: Columns connected by source and destination pairs yield an n-regular graph which can be n-colored.

colors so that each node has no two edges leaving it of the same color. Since each colored edge corresponds to a source and destination pair, we can assign each pair to the plane corresponding to its color. This permutation of points in a column ensures that sources and destinations end up in the same plane and that at most one source and one destination get assigned to each point in a given plane. □

Lemma 3.2.1 now follows as a corollary of Lemma 3.2.2:

Proof of Lemma 3.2.1: The idea here is to route all of the source-destination pairs into a sub-cube of G of size $\lceil m^{\frac{1}{d}} \rceil^d$, and then apply Lemma 3.2.2. Hence it suffices to show that the sources and destination can be packed into such a sub-cube using each edge of G at most $2\lceil m^{\frac{1}{d}} \rceil$ times. Again pick a dimension and consider the n^{d-1} planes connected by columns along that dimension. Since there are only m point to go around, at most $m^{\frac{1}{d}}$ planes contain more than $\lceil m^{\frac{1}{d}} \rceil^{d-1}$ points. Call a plane with more than that number *heavy*. Clearly, each heavy plane contains a

point in a column which is at most $\lceil m^{\frac{1}{d}} \rceil$ away from a light plane with an empty space in that column. Route this point along its column to that empty space and proceed in this manner until there are no heavy planes. Since all routing occurs along columns, and each column intersects at most $\lceil m^{\frac{1}{d}} \rceil$ heavy planes, no edge gets used more than $\lceil m^{\frac{1}{d}} \rceil$ times. Now consider the sub-cube of size $\lceil m^{\frac{1}{d}} \rceil^d$ chopped up into n non-overlapping slices, corresponding to the points in each of the n planes. By induction we can pack the points of each plane into the right positions indicated by the plane's corresponding slice using each edge of the plane at most $2\lceil (\lceil m^{\frac{1}{d}} \rceil^{d-1})^{\frac{1}{d-1}} \rceil$ $= 2\lceil m^{\frac{1}{d}} \rceil$ times. Then squash down along the columns to pack all of the slices into the sub-cube. We have now used each edge in the column dimension at most $2\lceil m^{\frac{1}{d}} \rceil$ times. □

By letting d vary from 1 to $\log N$, and observing that the edges of the d-dimensional grid can be divided into $2d$ matchings, Lemma 3.2.1 and the proof of Theorem 1.2.1 indicate that any embedding of a d-dimensional grid in the plane is within an $O(dN^{\frac{1}{d}})$ factor of optimal.

3.3 Finding Dense Subsets of Points in the Plane

Let $f_{opt} : G \rightarrow P$ be an optimal cost embedding of a d-dimensional grid among the N points in the plane. We wish to prove that given any K point subset $S \subset P$, at least half the points of S are contained in a dense sub-region of the plane. For our purposes *dense* will be defined in terms of $f_{opt}(G)$ and K by the following lemma:

Lemma 3.3.1 *Given any fixed K point subset of the N points in the plane, there exists a rectangular sub-region of the plane containing $\frac{K}{2}$ of these points with longest side length R, where $R \leq O\left(\frac{\|f_{opt}(G)\|}{K^{\frac{d-1}{d}}}\right)$.*

Proof: Consider the $s \times s$ square region containing all of the points. Starting at the left side of this region, move a vertical line to the right until exactly $\frac{K}{8}$ among the K

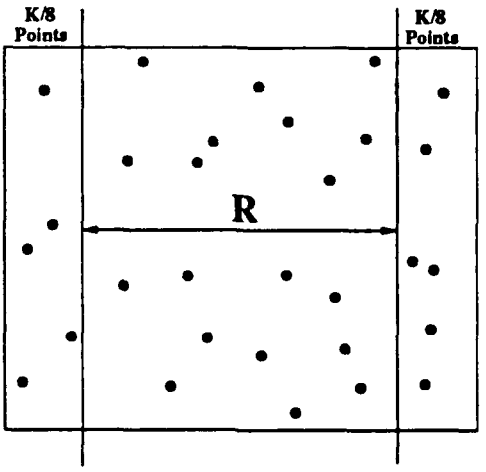

Figure 3-2: Finding dense subsets of points in the plane.

distinguished points are contained in the left region of the square defined by the line. Do the same with a vertical line working from the right to the left. Now let R be the distance between these two vertical lines. See Figure 3-2. Label the points in the left section from 1 to $\frac{K}{8}$. Do the same for the right section. By Lemma 3.2.1 we can draw paths in the optimal d-dimensional grid connecting corresponding points so that no edge of the optimal grid is used more than $O(K^{\frac{1}{d}})$ times. Each of these paths must cross distance R. Hence $\|f_{opt}(G)\| \geq \Omega\left(\frac{RK}{8K^{\frac{1}{d}}}\right)$ and therefore $R \leq O\left(\frac{\|f_{opt}(G)\|}{K^{\frac{d-1}{d}}}\right)$. Now do the same procedure with horizontal lines from the top and bottom. The center rectangle now has at least $\frac{K}{2}$ points and we may shrink it slightly until it holds exactly $\frac{K}{2}$. □

Our embedding procedure will be as follows. Use Lemma 3.3.1 to isolate $\frac{N}{2}$ points in a dense region S_1. Then embed an $\frac{N}{2}$-node piece of the grid G_1 in these points. Of the remaining points, use Lemma 3.3.1 to isolate $\frac{N}{4}$ in another perhaps slightly less

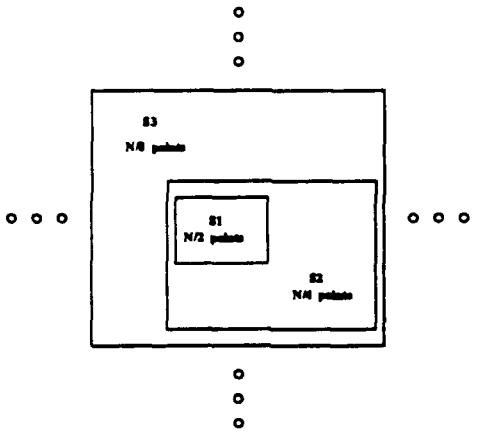

Figure 3-3: G_i embeds in the points contained in $S_i \setminus S_{i-1}$.

dense, region S_2. Then embed an $\frac{N}{4}$-node piece of the grid G_2 in these points. Proceed in this manner until all of the points in the plane are used up. Since each rectangular region S_i is obtained by sliding vertical and horizontal lines in from the boundary of the plane until the right number of points are captured, $S_1 \subset S_2 \subset \cdots \subset S_{\log N+1}$. G_i will be embedded in the $\frac{N}{2^i}$ points contained in $S_i \setminus S_{i-1}$. See Figure 3-3.

In this manner, we need a strategy for dividing G up into $\log N + 1$ pieces of geometrically decreasing size. Let $G = G_1 \cup G_2 \cup \cdots \cup G_{\log N+1}$ where G_1 is obtained by chopping G in half along one dimension, and each G_i is obtained by chopping G_{i-1} in half along its longest dimension. We call this the *geometric decomposition* of G. Figure 3-4 shows the geometric decomposition of the two dimensional grid. It is not hard to see that in d dimensions, this decomposition has the following property:

Observation 3.3.2 *Let G be a d-dimensional grid on N points. Then the geometric decomposition of $G = G_1 \cup G_2 \cup \cdots \cup G_{\log N+1}$ has the property that at most $\frac{1}{2^{j-i-1}}(\frac{N}{2^i})^{\frac{d-1}{d}}$ edges of G connect G_i to G_j for $j > i$ and for $j > i + d$ no edges connect G_i and G_j.*

Proof: This proof requires that we give a formal definition of the geometric decomposition of the grid in d dimensions. Label the pieces in the geometric decomposition $G = G_1 \cup G_2 \cup \cdots \cup G_{\log N+1}$. Imagine that the d-dimensional grid composed of the

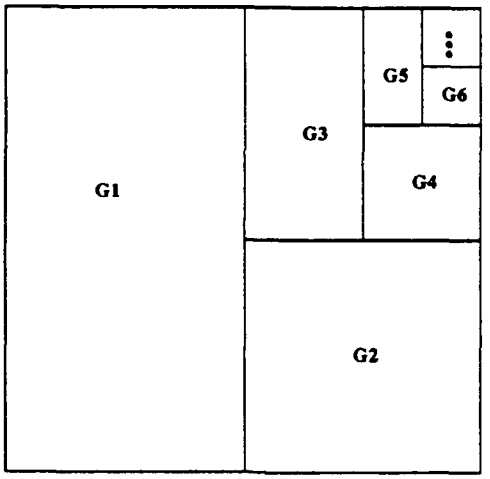

Figure 3-4: A geometric decomposition of the grid.

integer coordinate points in euclidean d-space:

$$G = \underbrace{\{1,2,\ldots,n\} \times \cdots \times \{1,2,\ldots,n\}}_{d}$$

Edges connect adjacent points in the grid whose coordinates differ along only one dimension. Then we have:

$$\begin{aligned}
G_1 &= [1,\frac{n}{2}] \times [1,n] \times \cdots \times [1,n] \\
G_2 &= [\frac{n}{2}+1,n] \times [1,\frac{n}{2}] \times [1,n] \times \cdots \times [1,n] \\
&\vdots \\
G_d &= [\frac{n}{2}+1,n] \times \cdots \times [\frac{n}{2}+1,n] \times [1,\frac{n}{2}] \\
G_{d+1} &= [\frac{n}{2}+1,\frac{3n}{4}] \times [\frac{n}{2}+1,n] \times \cdots \times [\frac{n}{2}+1,n] \\
G_{d+2} &= [\frac{3n}{4}+1,n] \times [\frac{n}{2}+1,\frac{3n}{4}] \times [\frac{n}{2}+1,n] \times \cdots \times [\frac{n}{2}+1,n] \\
&\vdots \\
G_i &= \underbrace{[\frac{2^{\lfloor \frac{i}{d} \rfloor+1}-1}{2^{\lfloor \frac{i}{d} \rfloor+1}}n+1,n] \times \cdots \times [\frac{2^{\lfloor \frac{i}{d} \rfloor+1}-1}{2^{\lfloor \frac{i}{d} \rfloor+1}}n+1,n]}_{i \bmod d}
\end{aligned}$$

$$\times \left[\frac{2^{\lfloor \frac{i}{d} \rfloor}-1}{2^{\lfloor \frac{i}{d} \rfloor}}n+1, \frac{2^{\lfloor \frac{i}{d} \rfloor+1}-1}{2^{\lfloor \frac{i}{d} \rfloor+1}}n\right]$$
$$\times \underbrace{\left[\frac{2^{\lfloor \frac{i}{d} \rfloor}-1}{2^{\lfloor \frac{i}{d} \rfloor}}n+1, n\right] \times \cdots \times \left[\frac{2^{\lfloor \frac{i}{d} \rfloor}-1}{2^{\lfloor \frac{i}{d} \rfloor}}n+1, n\right]}_{(d-1)-(i \bmod d)}$$

The edges of G which leave G_i and go into smaller G_j for $j > i$ all connect points along the $((i-1) \bmod d) + 1$ dimension. Notice that for each progressively smaller piece, $G_{i+1}, G_{i+2}, \ldots$, the size of the face adjacent to G_i is halved. Since $(\frac{N}{2^i})^{\frac{d-1}{d}}$ is an upper bound on the number of edges leaving G_i and entering G_{i+1}, the first part of Observation 3.3.2 follows. Furthermore, if we look at the ranges of coordinate values in the formula for G_i, we see that by level G_{i+d}, all of these ranges have been reduced in half, and hence the edges leaving G_i cannot possibly enter any of the G_j for $j > i + d$. □

We now indicate how to embed G_i into $S_i \setminus S_{i-1}$ using a divide and conquer subroutine. For simplicity's sake, we suppose that G_i is an M-node d-dimensional grid. It will be clear how to modify the subroutine to handle the case where G_i is not perfectly square.

3.3.1 The Divide and Conquer Subroutine

Let R_i be the length of the longest side of the region S_i containing the M-point subset $P_i \subset P$ into which we are embedding G_i. A straightforward method of embedding the d-dimensional grid G_i into P_i consists of dividing the points in half, recursively embedding half of G_i on each set of points, and then drawing in the $M^{\frac{d-1}{d}}$ connecting edges. If we are careful about how we divide up P_i, we can achieve an approximation algorithm which yields an embedding with cost bounded by a function of M and R_i. In order to implement such a divide and conquer algorithm, we use the *natural decomposition tree* for the grid. In two dimensions, we simply split G_i down the middle to get two rectangles. Then split these rectangle down the middle to get two squares, proceeding in this manner for $\log M$ levels. See Figure 3-5. This idea generalizes

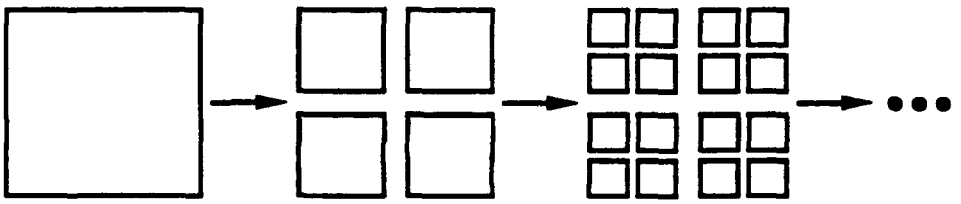

Figure 3-5: The natural decomposition tree for the grid is used in a divide and conquer embedding.

to d dimensions in the obvious fashion. The important observation concerning this decomposition is:

Observation 3.3.3 *Two adjacent blocks on the i^th level of the natural decomposition tree of G have at most $O\left(\left(\frac{N}{2^i}\right)^{\frac{d-1}{d}}\right)$ edges of G_i passing between them.*

In order to prove Observation 3.3.3, we give a formal definition of the natural decomposition tree in d dimensions. Let G be an N node d-dimensional grid. We define the standard binary decomposition tree on G by recursively splitting subsections of G in half. For example, if $N = n^d$ and G is the d-dimensional grid composed of the integer coordinate points in euclidean d-space:

$$G = \underbrace{\{1,2,\ldots,n\} \times \cdots \times \{1,2,\ldots,n\}}_{d}$$

At the first level we split G into two rectangular sub-grids isomorphic to

$$\underbrace{\{1,2,\ldots,\frac{n}{2}\} \times \{1,2,\ldots,n\} \times \cdots \times \{1,2,\ldots,n\}}_{d}$$

In this manner, at the i^{th} level, G has been decomposed into 2^i rectangular grids, all isomorphic to

$$\underbrace{\{1,2,\ldots,\frac{n}{2^{\lceil \frac{i}{d} \rceil}}\} \times \cdots \times \{1,2,\ldots,\frac{n}{2^{\lceil \frac{i}{d} \rceil}}\}}_{i \bmod d} \times \underbrace{\{1,2,\ldots,\frac{n}{2^{\lfloor \frac{i}{d} \rfloor}}\} \times \cdots \times \{1,2,\ldots,\frac{n}{2^{\lfloor \frac{i}{d} \rfloor}}\}}_{d-i \bmod d}$$

Proof of Observation 3.3.3: At level i, where blocks have size $\frac{N}{2^i}$, we split them by cutting along the $d-1^{st}$ dimensional plane perpendicular to the $i \bmod d+1$

dimension. Hence we cut $\left(\frac{n}{2^{\lfloor \frac{i}{d} \rfloor}}\right)^{d-1} \frac{1}{2^{i \bmod d}}$ edges.

$$\begin{aligned}
\left(\frac{n}{2^{\lfloor \frac{i}{d} \rfloor}}\right)^{d-1} \frac{1}{2^{i \bmod d}} &\leq n^{d-1} \frac{1}{2^{\lfloor \frac{i}{d} \rfloor (d-1) + i \bmod d}} \\
&= n^{d-1} \frac{1}{2^{i - \lfloor \frac{i}{d} + i \bmod d \rfloor}} \\
&\leq 2^{\frac{i}{d}} \frac{n^{d-1}}{2^i} \leq 2^{i(\frac{1}{d}-1)} n^{d-1} \\
&= (\frac{N}{2^i})^{\frac{d-1}{d}}.
\end{aligned}$$

□

We are almost ready to prove a bound on the cost of an embedding given by the divide and conquer algorithm. First, however, we need the following technical lemma:

Lemma 3.3.4 *Given N points in an $s \times s$ square region of the plane, for $0 \leq i \leq \log N$, we can subdivide the plane into disjoint rectangular regions $B_1^i, ..., B_{2^i}^i$, each containing $\frac{N}{2^i}$ points so that $\sum_{j=1}^{2^i} perimeter(B_j^i) \leq 4\sqrt{2^i}s$. Furthermore, $B_j^i = B_{2j-1}^{i+1} \cup B_{2j}^{i+1}$.*

Proof: To begin, assume the plane has been rotated so that no two points share the same vertical or horizontal coordinates. We now proceed by induction. Let B_1^0 = the $s \times s$ square containing all the points. Then $perimeter(B_1^0) = 4s$. On the even levels, we will divide boxes by drawing horizontal lines, and on the odd levels, by drawing vertical lines. So suppose that $B_1^{2i}, ..., B_{2^{2i}}^{2i}$ have been constructed to satisfy the lemma. Then we divide B_j^{2i} in half by drawing a vertical line which splits the points contained in B_j^{2i} into two equal sets contained in boxes B_{2j-1}^{2i+1} and B_{2j}^{2i+1}. Repeat this process on these two boxes, using horizontal lines, in order to generate $B_{4j-3}^{2(i+1)}, B_{4j-2}^{2(i+1)}, B_{4j-1}^{2(i+1)}$, and $B_{4j}^{2(i+1)}$. See Figure 3-6. We then have: $\sum_{j=1}^{2^{2(i+1)}} perimeter(B_j^{2(i+1)}) = \sum_{j=1}^{2^{2i}} \sum_{k=0}^{3} perimeter(B_{4j-k}^{2(i+1)}) \leq \sum_{j=1}^{2^{2i}} 2perimeter(B_j^{2i}) \leq 2(4\sqrt{2^{2i}}s) = 4\sqrt{2^{2(i+1)}}s$. The lemma follows by induction.

□

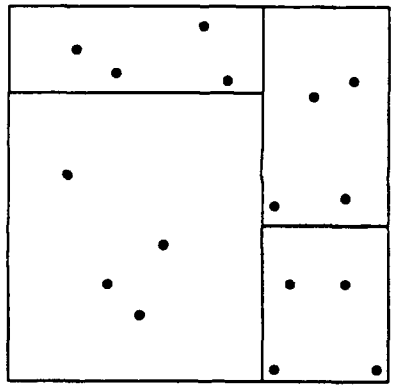

Figure 3-6: Dividing B_k^{j-2} into four rectangles containing an equal number of points

The following lemma shows that this divide and conquer subroutine gives a bound on the cost of embedding G_i into P_i in terms of M and R_i.

Lemma 3.3.5 *Suppose we are given M points in region S_i with longest side length R_i. Then in $O(M \log M)$ time we can embed the d-dimensional grid G_i on these points costing $O(\sqrt{M} \log M R_i)$ when $d = 2$ and $O(M^{\frac{d-1}{d}} R_i)$ when $d > 2$.*

Proof: As in the proof of Lemma 3.3.4, divide S_i into regions B_j^i, $0 \leq i \leq \log M$, $1 \leq j \leq 2^i$. Contained within the rectangular regions B_k^j at level j, we draw in the edges from the $(j+2)^{nd}$ level of the natural decomposition tree of G_i indicated in Figure 3-5. For any rectangular region B_k^j, the maximum distance between any two points is at most half the perimeter. Hence by Observation 3.3.3, drawing in all of the edges at level j will cost at most $(\frac{M}{2^j})^{\frac{d-1}{d}} \sum_{k=1}^{2^j} diam(B_k^j) \leq (\frac{M}{2^j})^{\frac{d-1}{d}} 2\sqrt{2^j} R_i = 2M^{\frac{d-1}{d}} 2^{(\frac{1}{2} - \frac{d-1}{d})j} R_i$

$$\leq \begin{cases} 2\sqrt{M} R_i & \text{if d=2} \\ 2M^{\frac{d-1}{d}} (2^{-\frac{1}{6}})^j R_i & \text{if d>2} \end{cases}$$

Summing over the $\log M$ levels gives us the lemma. In the $d = 2$ case, the sequence is not geometric so we get the extra log factor. We leave the running time analysis

to Section 3.4. □

We are now ready to give the main algorithm for embedding the entire d-dimensional grid G into P.

3.3.2 The Grid Embedding Algorithm

1. Use Lemma 3.3.1 to draw $\log N$ nested rectangular regions $S_1 \subset S_2 \subset \cdots \subset S_{\log N+1}$ in the plane so that S_1 contains $\frac{N}{2}$ points of P, $S_i \setminus S_{i-1}$ contains $\frac{N}{2^i}$ points of P, and $diameter(S_i) = O(\|f_{opt}(G)\|(\frac{2^i}{N})^{\frac{d-1}{d}})$. See Figure 3-3.

2. For $i = 1, \cdots, \log N + 1$ use the divide and conquer subroutine to embed piece G_i from the geometric decomposition into $S_i \setminus S_{i-1}$.

This algorithm gives an assignment of the nodes of G to the points of P. Call this map $h : G \rightarrow P$. The bounds on the diameters of the nested rectangles $S_1 \subset S_2 \subset \cdots \subset S_{\log N+1}$ allow us to prove the following lemma.

Lemma 3.3.6 *Given N points in the plane and $G = G_1 \cup G_2 \cup \cdots \cup G_{\log N+1}$, a geometric decomposition of the d-dimensional grid, we can embed the pieces G_1, ..., $G_{\log N+1}$ in the plane so that each G_i lies inside S_i and the total cost of these embeddings is $O(\|f_{opt}(G)\| \log^2 N)$ for $d = 2$ and $O(\|f_{opt}(G)\| \log N)$ for $d > 2$.*

Proof: Draw the nested rectangles as in Figure 3-3. Now apply the divide and conquer algorithm to embed G_i on the points in $S_i \setminus S_{i-1}$. By Lemma 3.3.5, the cost of embedding G_i is $O((\frac{N}{2^i})^{\frac{d-1}{d}} diameter(S_i))$ when $d > 2$ and $O(\sqrt{\frac{N}{2^i}} \log \frac{N}{2^i} diameter(S_i))$ when $d = 2$. Since $diameter(S_i) = O(\|f_{opt}(G)\|(\frac{2^i}{N})^{\frac{d-1}{d}})$ we have $\|h(G_i)\| = O(\|f_{opt}(G)\|)$ if $d > 2$ and $\|h(G_i)\| = (\|f_{opt}(G)\| \log N)$ if $d = 2$. Summing over the $\log N$ levels yields Lemma 3.3.6. □

All that now remains for us to do is account for the cost of the edges hooking together the $h(G_i)$'s and we will have an upper bound on the cost of the embedding $h(G)$. We put in these connecting edges in $\log N$ steps. The i^{th} step consists of putting in all

edges which extend from G_i to G_j where $j > i$. These edges are contained in S_j and by Observation 3.3.2 there are $O(\frac{1}{2^{j-i}}(\frac{N}{2^i})^{\frac{d-1}{d}})$ of them. The cost of connecting G_i to G_j is then $O\left(\frac{1}{2^{j-i}}(\frac{N}{2^i})^{\frac{d-1}{d}}\frac{\|f_{opt}(G)\|}{(N/2^j)^{\frac{d-1}{d}}}\right) = O\left(\frac{(2^{j-i})^{\frac{d-1}{d}}}{2^{j-i}}\|f_{opt}(G)\|\right) = O\left(2^{(1-\frac{d-1}{d})(j-i)}\|f_{opt}(G)\|\right)$. For $j = i, \cdots, i+d$ this sequence is geometrically decreasing when $d > 2$. Hence, the total cost of the i^{th} level interconnecting edges is $O(\|f_{opt}(G)\|)$. Summing over all i levels, we see that these interconnect edges add at most $O(\|f_{opt}(G)\| \log N)$ to the total cost. This finishes the proof of the most fundamental result in Part I: Theorem 1.2.2.

3.4 Running Time Analysis

We can implement an algorithm to find the nested rectangles by drawing vertical and horizontal lines to remove the required fraction of points from the margins and then incrementally shrinking the size of the resulting rectangle to capture the exact number of points needed. This will require us to sort the points once by x coordinate and once by y coordinate. After sorting, each region S_i can be found in $O(\frac{N}{2^i})$ time. Hence we can compute $S_1 \subset S_2 \subset \cdots \subset S_{\log N+1}$ in $O(N \log N)$ time. Computing the map $h : G \rightarrow P$ is accomplished by calling the divide and conquer subroutine to map each G_i into S_i. Let $T(M_i)$ be the running time for this subroutine on the M_i-node grid G_i. If we start by sorting the M_i points by x coordinate and by y coordinate, at the j^{th} level we can then split each region B_k^j in $O(\frac{M_i}{2^j})$ time. Hence, if we assume the M_i points are first properly sorted: $T(M_i) = O(M_i) + 4T(\frac{M_i}{4}) = O(M_i \log M_i)$. Adding in the initial sorting time, we still have $T(M_i) = O(M_i \log M_i)$. Then summing over all G_i, where $M_i = \frac{N}{2^i}$, we see that h can be computed in $O(N \log N)$ time.

Chapter 4

Arbitrary Weighted Graphs

We now turn to the case where we wish to embed an arbitrary weighted graph G in R^d. As indicated, we are able to give polynomial time approximation algorithms for this problem in the cases where P is an array of points (Theorem 1.2.3) or a randomly chosen uniform distribution of points (Theorem 1.2.4).

4.1 Embedding into an Array of Points

This section provides the proof of Theorem 1.2.3.

Theorem 1.2.3 *There exists a polynomial time algorithm which, when given an arbitrary N-node weighted graph G and a d-dimensional array of N points P in R^d, embeds G in P with cost within an $O(\log^2 N)$ factor of optimal.*

In this section, we assume that the points P are arranged in an evenly spaced d-dimensional square array in R^d. Let $G = (V, E)$ be an arbitrary weighted N-node graph with edge weights $w(e)$ for each $e \in E$. A $\frac{1}{k} - \frac{k-1}{k}$ *separator* of G is a decomposition of V into disjoint sets U and W such that $min(\|U\|, \|W\|) \geq \frac{N}{k}$. Let S be the set of edges that join points in U to points in W. Then we define the cost of the separator to be $\|S\| = \sum_{e \in S} w(e)$. Leighton and Rao [38] have recently given

an approximation algorithm for finding minimum cost separators. We will use the following result from their paper.

Theorem 4.1.1 (Leighton, Rao) *There exists a polynomial time algorithm SEP such that given a weighted, undirected graph G, SEP finds a $\frac{1}{10} - \frac{9}{10}$ separator for G which has cost less than $O(\log N)$ times the optimal cost $\frac{1}{8} - \frac{7}{8}$ separator of G.*

We are now in a position to prove Theorem 1.2.3. We give the proof for the case when P is a 2-dimensional grid of points. It is not hard to extend the ideas to handle the general case where the dimension d is a constant greater than 2. Let OPT be the set of edges crossing the optimal $\frac{1}{8} - \frac{7}{8}$ separator.

We proceed with a divide and conquer assignment of points in G to the square array P. Consider the optimal embedding $f_{opt} : G \rightarrow P$. Let s be the side length of the array P.

Claim 4.1.2 $s = O(\frac{\|f_{opt}(G)\|}{\|OPT\|})$

Proof: Move a line perpendicular to one dimension of P from left to right across the plane, until it splits off the leftmost $\frac{1}{8}$ of the nodes in P. From this starting point, continue to move the line across the array until it splits off $\frac{7}{8}$ of the nodes. At each point in its traversal, the line cuts $\geq \|OPT\|$ edge weight from $f_{opt}(G)$. Since the line moves distance $\frac{3s}{4}$, $\|f_{opt}(G)\| \geq \frac{3s}{4}\|OPT\|$. □

Now use SEP from Theorem 4.1.1 to obtain a $\frac{1}{10} - \frac{9}{10}$ separator $V = U \cup W$ of G which has cost within $O(\log N)$ times $\|OPT\|$. Let C be the set of edges joining U and W. We can now write $G = (U \cup W, E_1 \cup E_2 \cup C)$. Again, move a line from left to right across P and stop at the point where the line cuts off exactly $\|U\| \geq \frac{N}{10}$ leftmost points. (The line may contain a jog.) See Figure 4-1. Recursively embed (U, E_1) and (W, E_2) in the left and right sub-rectangles of P defined by the line. Notice that since at each level we are splitting sub-graphs of G with $\frac{1}{10} - \frac{9}{10}$ separators, as long as we cut the sub-rectangles of P along their longest side, the aspect ratio of any rectangle never exceeds 10 to 1. We measure the cost of this recursively defined embedding by

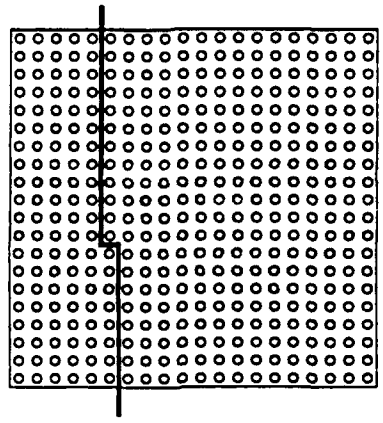

Figure 4-1: Embedding G into a grid of points.

adding up the cost of the edges crossing cuts at each level. At the top level, the cost of embedding C in P is bounded above by $\|C\|s \leq \|f_{opt}(G)\| \log N$. To bound the costs introduced at the lower levels, we will need a slightly more general claim.

Claim 4.1.3 *Let s be the length of the longest side of an M-node sub-rectangle R of P with aspect ratio bounded by some constant k. Then if (U, E_1) is* any *M-node subgraph of G with optimal $\frac{1}{8} - \frac{7}{8}$ separator OPT, $s = O(\frac{\|f_{opt}(U)\|}{\|OPT\|})$.*

Proof: Move a line from left to right across P, starting where the line first splits off $\frac{M}{8}$ of the points *of* $f_{opt}(U)$ and stopping when it splits off the leftmost $\frac{7M}{8}$ points. Let the distance between these two cuts be l_1. Repeat this process with a horizontal line moving from the top to bottom of P, and let the distance measured this way be l_2. Then $\geq \frac{M}{2}$ of the points of $f_{opt}(U)$ are contained in an $l_1 \times l_2$ region. Hence since R is roughly square, $s = O(max(l_1, l_2))$. Now by arguments similar to those given above, $\|f_{opt}(U)\| \geq l_i \|OPT\|$ for $i = 1, 2$. It follows that $s = O(\frac{\|f_{opt}(U)\|}{\|OPT\|})$. □

To finish the proof of Theorem 1.2.3 consider the i^{th} level of recursion in our embedding algorithm. Let $P = P_1 \cup \cdots \cup P_k$ be the disjoint union of the points contained in the $k = 2^i$ rectangular regions defined at this level. Let P_i have longest side length s_i. Then let $V = U_1 \cup \cdots \cup U_k$ be the disjoint union such that U_i is

embedded in P_i. Let OPT_i be the optimal $\frac{1}{8} - \frac{7}{8}$ separator of the subgraph of G determined by U_i, and let C_i be the edges crossing the $\frac{1}{10} - \frac{9}{10}$ separator found by SEP. Then the cost of the embedding introduced at level i is bounded above by $\sum_{i=1}^{k} s_i \|C_i\| \leq \sum_{i=1}^{k} s_i \|OPT_i\| O(\log N)$. By Claim 4.1.3 this sum is bounded above by $\sum_{i=1}^{k} \|f_{opt}(U_i)\| O(\log N) \leq \|f_{opt}(G)\| O(\log N)$. Summing over all $\log N$ levels of the recursion, the total cost of the embedding is $\|f_{opt}(G)\| O(\log^2 N)$.

4.2 Embedding into a Randomly Distributed Set of Points

This section presents the proof of Theorem 1.2.4.

Theorem 1.2.4 *There exists a polynomial time algorithm which, when given an arbitrary N-node graph G and a random set of N uniformly distributed points P in R^d, with high probability embeds G in P with cost within an $O(\log^2 N)$ factor of optimal.*

An algorithm similar to the one described above achieves an $O(\log^2 N)$ times optimal embedding with high probability when the points P are randomly distributed in a square $s \times s$ planar region S. Again, we use SEP to find a good $\frac{1}{10} - \frac{9}{10}$ separator of G, split the planar region into appropriately sized pieces with a vertical line, and proceed recursively to embed the two sub-graphs of G in these pieces. It is not hard to show that with high probability Claim 4.1.2 still holds in the randomly distributed case. Hence, the embedding cost generated by the cut at the top level of the algorithm with high probability is $O(\|f_{opt}(G)\| \log N)$.

All that is needed at this point is a probabilistic analog of Claim 4.1.3. So suppose we are at the i^{th} level of recursion, and we are embedding some M-node subgraph (U, E_1) of G into a sub-rectangle R of the plane with longest side length r. As in Claim 4.1.3, consider the $\frac{M}{2}$ points of $f_{opt}(U)$ contained in an $l_1 \times l_2$ region. It still

holds that $\|f_{opt}(U)\| \geq l_i\|OPT\|$ for $i = 1, 2$. We simply need to show that with high probability $r = O(max(l_1, l_2))$.

For this, we will need to prove some preliminary probabilistic results about uniform distributions of points in the plane. All of these results use standard Chernoff bound analysis. See Section 7.5.1 for a review of Chernoff bounds.

We call a sub-region R of the $s \times s$ square S an $r_1 \times r_2$ *sub-rectangle* if the sides of R are parallel to the sides of S and the lengths of its sides are r_1 and r_2. The expected number of points lying in R is $E_p(R) = \frac{Nr_1r_2}{s^2}$.

Lemma 4.2.1 *Fix l and w. Then there exists a constant λ such that for all N with high probability no sub-rectangle R with shortest side length r_1 such that $w \leq r_1 \leq 2w$, longest side r_2 such $l \leq r_2 \leq 2l$ and area $\geq \frac{\lambda s^2 \log N}{N}$ contains more than $14E_p(R)$ or less than $\frac{1}{18}E_p(R)$ points.*

The proof of Lemma 4.2.1 will use the following claim:

Claim 4.2.2 *Fix a region R in the square S and a constant c. Then there exists a constant λ such that for all N if the area of R is $\geq \frac{\lambda s^2 \log N}{N}$ then with probability $\geq 1 - N^{-c}$, R contains $\leq \frac{3}{2}E_p(R)$ or $\geq \frac{1}{2}E_p(R)$ points.*

Proof: Let q be the probability that a point lies in R. Then using well known Chernoff bounds [42] we know that Prob[R contains $\leq \frac{3}{2}E_p(R)$ or $\geq \frac{1}{2}E_p(R)$ points] $\leq e^{-\beta qN}$ for some constant β. Now we may pick λ sufficiently large so that $q \geq \frac{\alpha \log N}{N}$ where α is large enough so that $e^{-\alpha\beta \log N} \leq N^{-c}$. □

Proof of Lemma 4.2.1: The lemma is vacuously true if $lw \leq o(\frac{s^2 \log N}{N})$, so we may assume that $lw \geq \Omega(\frac{s^2 \log N}{N})$ and $\frac{s^2}{lw} \leq O(\frac{N}{\log N})$. We proceed to show that very few rectangular regions R are either so dense or so sparse that they violate the conditions of Lemma 4.2.1. We will handle the case where R contains more than $14E_p(R)$ points first. Divide S into $\frac{s^2}{lw}$ sub-regions of size $l \times w$ in two ways. One way, the length l side is horizontal, and the other way it is vertical. See Figure 4-2 Then

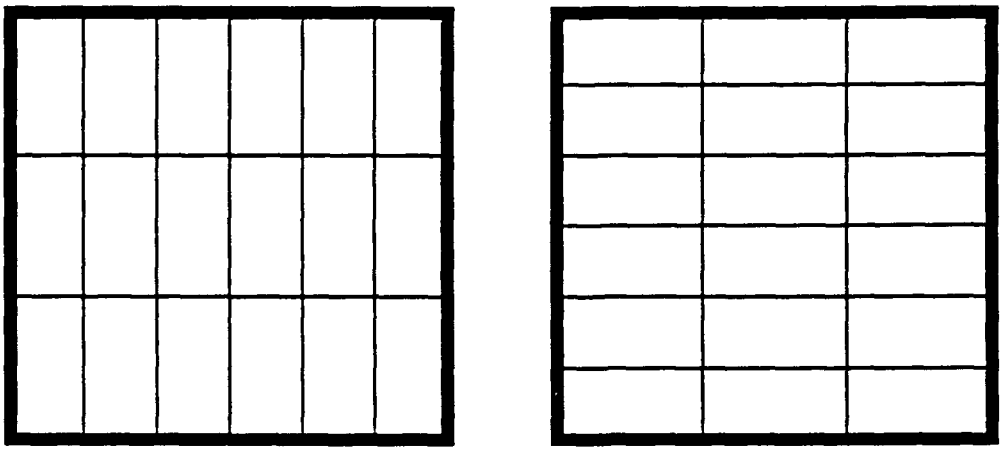

Figure 4-2: Divide the square region into $l \times w$ sub-rectangles in two ways.

R lies inside some 3×3 block of these sub-regions. There are less than $2\frac{s^2}{lw}$ of these blocks. So the probability that R contains $\geq 14E_p(R)$ points is $\leq 2\frac{s^2}{lw} \leq O(\frac{N}{\log N})$ times the probability that some fixed 3×3 block contains $\geq 14E_p(R)$ points. Let B be some such fixed block. But the expected number of points for such a block is $E' = 9\frac{lw}{s^2}N$. So $14E_p(R) \geq \frac{3}{2}E'$. But, Prob[number of points in $B \geq \frac{3}{2}E'$] $\leq N^{-c}$ by Claim 4.2.2. Hence Prob[some R contains $\geq 14E_p(R)$ points] $\leq N^{-(c-1)}$.

Now consider the case when R contains less than $\frac{1}{18}E_p(R)$ points. Divide S into $\frac{9s^2}{lw}$ $\frac{1}{3}l \times \frac{1}{3}w$ sub-regions. Then some sub-region lies inside of R. So the probability that R contains $\leq \frac{1}{18}E_p(R)$ points is $\leq$ the probability that some one of these small regions S' contains $\leq \frac{1}{18}\frac{Nr_1r_2}{s^2}$ points. But $E_p(S') = \frac{Nr_1r_2}{9s^2}$, hence this probability is $\leq$ the probability that S' contains less than $\frac{1}{2}E_p(S')$. Once again, by the above claim, the probability that a fixed S' contains $\leq \frac{1}{2}E_p(S')$ is $\leq N^{-c}$. Hence Prob[some R contains $\leq \frac{1}{18}E_p(R)$ points] $\leq N^{-(c-1)}$. □

We can now prove the stronger Lemma 4.2.3 which holds for any sufficiently large, bounded aspect ratio sub-rectangle:

Lemma 4.2.3 *There exist a constant λ such that with high probability no sub-*

rectangle with area $\geq \frac{\lambda s^2 \log N}{N}$ *contains more than* $14E_p(R)$ *or less than* $\frac{1}{18}E_p(R)$ *points.*

Proof: Let R be a sub-rectangle satisfying the assumptions of the lemma. Let r_1 and r_2 be the lengths of the shortest and longest sides of R. Divide the ranges of possible values for r_1 and r_2 up into sub-intervals of length $\frac{s}{N}$. Each pair of sub-intervals corresponds to a pair of values for l and w in Lemma 4.2.1. There are at most N^2 such possible pairs of values. Notice in the proof above that λ does not depend on l or w, so we may pick some constant λ to suffice for all and yield probability $\leq N^{-(c+2)}$. Since there are $< N^2$ sub-interval pairs, the probability that any R violates the conclusion of Lemma 4.2.3 is $\leq N^{-c}$. □

Lemma 4.2.4 *With high probability, any sub-rectangle R generated during our recursive embedding procedure has aspect ratio bounded above by some constant k whenever R contains* $\geq \lambda \log N$ *points.*

Proof: We proceed by induction on the construction of rectangles. Lets say we start with some rectangle R' which has aspect ratio $\leq k$ and contains M points of the distribution within its area. We are going to split its points better than $\frac{1}{10} - \frac{9}{10}$ with a line perpendicular to its longest side. What is the probability that $\frac{1}{10}$ of its points are squashed up into some sub-rectangle R with aspect ratio $\geq k$? See Figure 4-3. When such a squashing occurs, we have $\geq \frac{1}{10}$ of the points in R' packed into $\leq \frac{1}{k}$ of the area of R'. Let $E_p(R) = N\frac{Area(R)}{s^2}$ be the expected number of points in the rectangle R. From Lemma 4.2.3 we know that with high probability R' has area $\leq 18\frac{M}{N}s^2$. This implies that there are $\frac{M}{10}$ points in a sub-rectangle of size $\frac{18}{k}\frac{M}{N}s^2$. For $k \geq (18)(10)(14)$ this violates Lemma 4.2.3. □

We now return to our original goal: proving that the recursive embedding algorithm using SEP generates near optimal embeddings with high probability given a uniform distribution of points. We left off at the i^{th} level of recursion. At this level, we wish to embed some M-node subgraph (U, E_1) of G into a sub-rectangle R of the

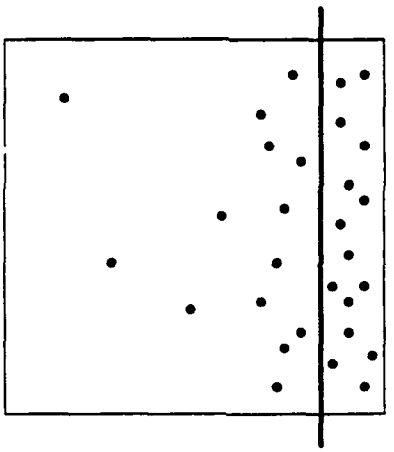

Figure 4-3: A bad distribution of points leads to high aspect ratio rectangles.

plane with longest side length r. The $\frac{M}{2}$ points of $f_{opt}(U)$ are contained in an $l_1 \times l_2$ region R. We need to show that with high probability $r = O(max(l_1, l_2))$. Now if it is not true that $r = O(max(l_1, l_2))$, then R is a sparsely populated rectangular region with area $\geq \frac{1}{k}r^2 >> l_1 l_2$ which contains only M points. But the $l_1 l_2$ region also contains $\geq \frac{M}{2}$ points. When $M \geq \lambda \log N$ these conditions violate Lemma 4.2.3.

Now we must handle the case where $M \leq \lambda \log N$. Let us alter the algorithm used for embedding arbitrary graphs in arrays so that when regions reach size $\leq \lambda \log N$ we stop. Then we have sub-divided $V = U_1 \cup \cdots \cup U_k$ such that $\|U_i\| \leq \lambda \log N$. Furthermore, each U_i is contained in a region with diameter $\leq O\left(\frac{\|f_{opt}(U_i)\|}{\|OPT_i\|}\right) \leq O(\|f_{opt}(U_i)\|)$. Hence we can embed the remaining subgraphs into their regions any way we want with cost $\leq \sum_i \|f_{opt}(U_i)\| \lambda^2 \log^2 N \leq O(\|f_{opt}(G)\| \log^2 N)$. Therefore, with high probability, the total cost of the embedding is $O(\|f_{opt}(G)\| \log^2 N)$. □

Chapter 5

Applications to Parallel Processing

Several important practical problems in parallel processing concern the efficient embedding of large computational problems into parallel architectures. We consider a distributed model of parallel computing where a parallel algorithm is composed of a set of processes. Each process performs a certain set of computations and sends and receives data from other processes. To avoid confusion of processes with processors, we will sometimes refer to processes as tasks. We define the *computation graph* for an algorithm to be the graph which has one node for each process and edges between nodes whose processes exchange data. In some cases, we may consider the *weighted computation graph* in which edge weights represent the relative amount of communication carried by each edge. Examples of algorithms and their corresponding computation graphs include:

1. Circuit simulation. Each node of the computation represents a device in the circuit with edges between devices which exchange electrical information.

2. Numerical methods for solving differential equations. Various finite difference methods yield grids, cubes, and other computation graphs for numerically solv-

ing differential equations.

3. Fast Fourier transform. The natural computation graph is the butterfly.

A problem which arises naturally in this context concerns how to assign an algorithm's tasks to a machine's processors and efficiently embed the computation graphs for these algorithms into the parallel architecture. Our geometric embedding results address this problem by providing algorithms for assigning tasks to processors for a large class of machines: array-based parallel processors.

5.1 Minimizing Communication Load

We measure the efficiency of an embedding in a parallel architecture in terms of the communication load which that embedding induces on the interconnect network. Communication load is the total volume of inter-processor communication carried by the interconnect network on the architecture. For any given parallel algorithm, this load may vary dramatically depending upon how tasks are assigned to processors. Let $m_1, \ldots, m_k$ be the messages which must be exchanged between processors during the algorithm. Then if $d(m)$ is the distance traveled by a message through the interconnection network, the total communication load is simply: $\sum_i d(m_i)$. In this manner, communication load is a measure of the total volume of traffic the network will be required to bear during the algorithm.

As an illustration, consider a parallel processing problem represented as a graph on a set of tasks: $t_1, \ldots, t_k$. The weight of an edge in this graph represents the amount of communication required between tasks t_i and t_j. See Figure 5-1. We need to optimally assign these tasks to processors in a manner which minimizes the communication load on the network. For example, if we are interested in running a circuit simulation, then each task t_i would correspond to a device in the circuit and we would need to minimize communication load induced by simulating the electrical signals transmitted between devices.

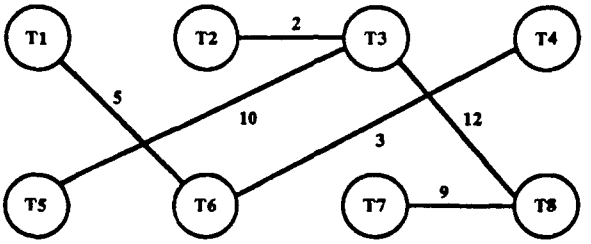

Figure 5-1: Computation graph for a problem to be solved on a parallel machine.

Our results on weighted graph embeddings allow us to give a provably near-optimal approximation algorithm for minimizing communication load on array-based parallel processors. Here we understand an array-based architecture to mean any N-node grid, cube, or higher dimensional array. As a corollary of Theorem 1.2.3 we can embed any weighted computation graph in an array-based machine minimizing communication load to within an $O(\log^2 N)$ factor of optimal.

Corollary 5.1.1 *There exists a polynomial time algorithm which, when given an N-processor array-based machine M and a parallel processing task consisting of a set of tasks $t_1, \ldots, t_N$ with communication costs $c(t_i, t_j)$ between tasks, can compute an assignment of tasks to the processors of M which achieves a communication load which is within $O(\log^2 N)$ times optimal.*

Proof: In a k-dimensional array-based processor, the distance between two processors p_1 and p_2 is $dist(p_1, p_2)$, where $dist$ is the Manhattan metric in k-space. Within constant factors depending on k, this metric is approximated by the euclidean distance function. Let $G = (V, E)$ be the complete graph on $V = t_1, \ldots, t_N$ with edge weights $c(t_i, t_j)$. Then we simply need to solve the weighted geometric embedding problem of mapping G into a k-dimensional grid. □

5.1.1 Simulating Other Architectures

Corollary 5.1.1 can also be interpreted as providing an algorithm for near-optimal simulations of other architectures on array-based machines. An alternate architecture can be represented as an N-node computation graph where nodes are processors and edge weights represent the amount of traffic typically present on the link between two processors. We can then use Corollary 5.1.1 to embed the alternate architecture in the array-based machine in a manner which minimizes communication load to within an $O(\log^2 N)$ factor of optimal.

5.2 Dynamic Allocation of Resources on a Multiprocessor

Suppose we are given an N-processor parallel machine configured as a graph $G = (V, E)$ where $\|V\| = N$, and a schedule of problems to process: $p_1, \ldots, p_k$. Each problem p_i requires a certain number n_i of processors. The machine's operating system dynamically allocates processors to problems. As problems finish, *holes* of idle processors open up in the network. How can we efficiently assign processors from the holes to the remaining problems in a manner which minimizes the communication load on the network? If each problem p_i corresponds to a computation graph c_i, this problem consists of finding optimal embeddings of c_i into the network holes. Using our results from Theorem 1.2.4 on embedding weighted graphs into randomly distributed points in euclidean space, we can show how to embed an arbitrary computation graph C on n nodes into randomly distributed subsets of an array processor within a $O(\log^2 n)$ factor of optimal.

Corollary 5.2.1 *There exists a polynomial time algorithm which, when given a k-dimensional array processor M with n randomly distributed idle processors and a computation graph C on n nodes, with high probability can assign nodes of C to the*

idle processors in a manner which minimizes the additional communication load on M to within an $O(\log^2 n)$ factor of optimal.

Given the large number of parallel algorithms devised for grids, cubes, hypercubes, butterflies, and shuffle exchange graphs, we are frequently interested in problems p, which have one of these graphs as their computation graph. For example, algorithms for solving differential equations frequently have computation graphs which are grids, cubes, or higher dimensional arrays. Fast Fourier transform and Batcher's merge sort run naturally on the butterfly. In these cases, we can drop the assumption that idle processors are randomly distributed, and give faster algorithms which always assign processors in a near-optimal manner. In fact, when C is a hypercube, shuffle-exchange graph, or butterfly, *any* processor assignment strategy is near-optimal.

Corollary 5.2.2 *Given a k-dimensional array processor M with n idle processors and a computation graph C on n nodes which is a uniformly weighted hypercube, butterfly, or shuffle exchange graph, then* any *assignment of the nodes of C to the idle processors creates additional communication load which is within an $O(\log n)$ factor of optimal.*

For grids, cubes, and higher dimensional arrays, our geometric embedding results yield very fast approximation algorithms for the processor assignment problem.

Corollary 5.2.3 *There exists an $O(n \log n)$ time algorithm which, when given a 2-dimensional array processor M with n idle processors and a computation graph C on n nodes which is an uniformly weighted, can assign the nodes of C to the idle processors to create additional communication load which is within an*

1. *$O(\log^2 n)$ factor of optimal when C is a grid.*

2. *$O(\log n)$ factor of optimal when C is a cube, or higher dimensional array.*

5.2.1 Wafer-Scale Integration and Reconfiguring Around Faults in an Array-Based Processor

Now suppose we are given an array-based parallel machine with a number of faulty nodes. Similarly, consider a silicon wafer with processors arranged in an interconnected grid where some of the processors are faulty. We assume a model where messages can still pass through a failed processor, but its processing capabilities are dead. Then the live nodes correspond to a set of points P in the plane or R^d. For arbitrary faults, Theorems 1.2.2 and 1.2.1 yield algorithms for reconstructing a grid, cube, hypercube, shuffle exchange graph, or butterfly on the live nodes with near-optimal communication load. On the other hand, if the faults are randomly distributed, then Theorem 1.2.4 yields an algorithm for embedding arbitrary architectures on the live nodes with near-optimal communication load.

5.3 Configuring Large Area Distributed Computing Networks

Suppose we need to link together a number of widely dispersed computing centers into a single network. For example, the weather service may need to configure a large number of weather stations around the country into a single network for weather prediction. We may wish to dynamically add and remove centers from this network as time progresses. For example, if the weather service were tracking and analyzing a storm, the stations in the network would be changing over time as the storm moved across the country. In the case of such large area distributed networks, communication costs between centers will comprise a significant percentage of the total cost of operating the network. If we assume that communications between centers will take place over existing telecommunications channels, the operator of the network will pay rates proportional to the distance traveled for each piece of communication. Hence

the operator would be interested in configuring the network so as to minimize the total embedded edge length. Assuming that centers lie on the plane, this reduces to a geometric embedding problem. Our results indicate how to produce near-optimal solutions to this problem when the network is a grid, cube, hypercube, shuffle-exchange graph, or butterfly. For other network topologies, we can achieve the same results provided the centers are uniformly distributed.

Part II

Query-Retrieval

Chapter 6

Introduction

6.1 Overview of Query-Retrieval Results

Query-retrieval problems have received considerable attention in the computational geometry literature [10, 13, 14, 15, 16, 18, 20, 24, 25]. Typically, a query-retrieval problem consists of a set of n geometric objects (e.g. points in the plane), and an unlimited sequence of queries (e.g. "of the n points in the plane, find the k points that are closest to position (x, y)"). The task is to preprocess the n objects and devise a data structure so that each query can be answered as quickly as possible.

The most important measures of efficiency in a query-retrieval problem are the space required by the data structure and the time needed to answer each query. For example, typical query-retrieval problems require $\Omega(n)$ space and $\Omega(k + \log n)$ time per query where k is the size of the response to the query. The algorithms and data structures described in this part of the thesis all achieve the optimal time bound and use $\Theta(n)$ or $\Theta(n \log n)$ space, depending on the problem being solved. Preprocessing time and space (used to construct the data structure) are also of concern, but are usually not as important as data structure space and query response time. All of the algorithms described in Part II of this thesis use polynomial preprocessing time and space. Using probabilistic methods, the preprocessing time and space can be reduced

to $O(n \log^2 n)$.

6.2 Main Results

In this part of the thesis, we describe a new technique for solving query-retrieval problems in optimal time with optimal or near-optimal space. The technique incorporates planar separators, filtering search, and the probabilistic method to compact k^{th}-order Voronoi diagrams (and/or other suitable proximity diagrams) from $k^{O(1)}n$ space to $O(n)$ space without losing any of the information that is essential for solving query problems. As examples, we use the technique to construct algorithms and data structures for k-nearest neighbor search, circular range search, and half-space range search. A brief description of each of these problems and of our results is provided below.

6.2.1 Planar k-Nearest Neighbor Search

Input: A set P of n points in the Euclidean plane E^2 and an integer $k \geq 0$.
Query: Find the k points in P that are closest to a query point q.

We show how to solve this problem using $O(n)$ space and $O(k + \log n)$ time per query, both of which are optimal. If k is not fixed, but is given as a part of the query (i.e., the query is a pair (q, k) with $q \in E^2$ and $k \leq n$), then our solution uses $O(n \log n)$ space and $O(k + \log n)$ time per query. The best previously known solution to this problem is due to Chazelle, Cole, Preparata, and Yap [15] who construct a data structure using $\Theta(n(\log n \log\log n)^2)$ space.

We also consider the k-nearest neighbors problem in some other non-euclidean metrics. For example, we show how to construct data structures and algorithms with equivalent performance in the power-distance metric and the additive-weight metric. This metrics are defined and described in Section 7.1.2. Here we simply point out that Voronoi diagrams in these alternative metrics are well known and have several

applications [6, 5, 7, 8, 22, 20, 41, 43], but the k-nearest neighbor searching problem in these metrics does not appear to have been studied previously. As an example of the applications of these metrics, we use our technique for computing k-nearest neighbors in the power-distance metric to solve the 3-dimensional half-space range search problem.

In the special case that each query point q belongs to a restricted set Q of $O(n)$ points (e.g., $Q = P$), the query response time of our algorithms can be improved to $O(k)$ without increasing the space. This restricted searching result could be useful in conjunction with heuristics for large traveling salesman problems that repeatedly compute the k nearest neighbors of various cities along the tour [34]. For small problem instances, it is possible to simply precompute the k-nearest neighbors of every node and store them, thereby using $O(kn)$ space. For large n, however, this approach is not possible, and the techniques described in Part II of this thesis become more appropriate. In this application, our approach uses $O(n)$ space and $O(k)$ query time, both of which are optimal.

6.2.2 Circular Range Search

Input: A set P of n points in the Euclidean plane E^2.

Query: Find all points of P contained in a disk in E^2 with radius r centered at q.

We show how to solve this problem using $O(n \log n)$ space and $O(k+\log n)$ time per query where k is the number of points in the disk. Several algorithms for this problem have been reported in the literature [10, 13, 15, 16]. The best previously known algorithm (due to Chazelle, Cole, Preparata, and Yap [15]) uses $\Theta(n(\log n \log\log n)^2)$ space and $O(k + \log n)$ time per query. We transform the circular range search problem into the planar k-nearest neighbors search problem and apply the filtering search technique given in [13] to obtain the result.

If the centers of the query disks are known ahead of time and if the number of centers is $O(n)$ then the query time can be improved to $O(k)$ without increasing the space.

6.2.3 Half-Space Range Search in Three Dimensions

Input: A set P of n points in Euclidean space E^3.

Query: Find all points of P that are contained in the half space defined by $ax + by + cz \leq d$.

We show how to solve this problem with $O(n \log n)$ space and $O(k + \log n)$ query time, where k is the number of points found to be contained in the half-space. Previously, Chazelle and Preparata [20] had given a data structure for this problem using $O(n \log^5 n \log\log^4 n)$ space which was later improved to $O(n \log^2 n \log\log n)$ space by Clarkson and Shor [24]. We transform the range search problem into the k-nearest neighbor search problem in the power-distance metric and use filtering search to obtain our solution.

6.3 Outline of Part II

The remainder of Part II is divided into three sections. Chapter 7 presents our compaction technique by constructing an $O(n)$-space data structure for k-nearest neighbor search in the Euclidean plane. Initially a randomized construction is given, and later (in Section 7.8) we show how to use the probabilistic method [42] to give a deterministic construction. We also indicate how the compaction technique can be applied to nearest neighbor search problems in other metrics. Section 7.10 briefly illustrates how we can build a linear data structure for k-nearest neighbor search with significantly improved preprocessing time by re-introducing some randomness into the deterministic construction. Finally, Chapter 8 discusses applications of the compaction technique to circular range search problems in the plane and half-space

range search problems in three dimensions. We conclude this chapter with a broad generalization of the technique and suggest applications to other retrieval problems.

Chapter 7

Voronoi Diagram Compaction

7.1 Review of Voronoi Diagrams

The importance of Voronoi Diagrams in computer science and other fields has long been recognized [1, 6, 36]. Here we give a brief overview of the properties of Voronoi diagrams in the euclidean plane and other metrics which are important to the results proved in this thesis. Throughout this section we let $P = p_1, \ldots, p_n$ be a set of points in the plane which are in general position.

7.1.1 The Euclidean Plane

For any pair $p_i, p_j \in P$, let $h(p_i, p_j)$ be the half-plane containing p_i which is defined by the perpendicular bisector between p_i and p_j. Then the Voronoi diagram defined by $p_1, \ldots, p_n$ is the partition of the plane into convex regions $R(p_1), \ldots, R(p_n)$ where $R(p_i) = \bigcap_{j \neq i} h(p_j, p_i)$. In this manner, the boundary edge between any two convex regions in the Voronoi diagram is a segment of a perpendicular bisector between two points p_i and p_j. $R(p_i)$ consists of those points in R^2 which are closer to p_i than any other point $p_j \in P$. Likewise, the k^{th}-order Voronoi diagram for $p_1, \ldots, p_n$ is the partition of the plane into convex regions $R(T)$ where $T = p_{i_1}, \ldots, p_{i_k}$ is any subset of k points and $R(T) = \bigcap_{p \in T, q \in P \setminus T} h(p, q)$. $R(T)$ consists of all points in

the plane whose k-nearest neighbors are exactly T. Similarly, the boundary edge between any two convex regions in the k^{th}-order Voronoi diagram is also a segment of a perpendicular bisector between two points p_i and p_j. This yields the following observation:

Observation 7.1.1 *Whenever $R(T)$ and $R(S)$ are adjacent faces in the k^{th}-order Voronoi diagram, T and S differ in exactly one point.*

Proof: Let s and t be the points whose perpendicular bisector defines the common edge between $R(T)$ and $R(S)$. WLOG suppose that $R(T) \subset h(t,s)$ and $R(S) \subset h(s,t)$. Then $s \in S$ and $t \in T$ so S and T differ in at least one point. Now suppose that S and T differ on another pair of points s' and t'. Then $R(S) \subset h(s,t) \bigcap h(s',t')$ and $R(T) \subset h(t,s) \bigcap h(t',s')$. But $h(s,t) \bigcap h(s',t')$ and $h(t,s) \bigcap h(t',s')$ do not share a common edge. We have a contradiction and S and T must differ in at most one point. □

Along with the above observation, this thesis uses the following well known properties of the k^{th}-order Voronoi diagram V:

1. V is a planar graph with $O(kn)$ edges and faces. [36]

2. V can be computed in $O(k^2n + n \log n)$ time. [1]

7.1.2 Other Metrics

In this thesis, we also apply the compaction technique to solve query-retrieval problems in other metric spaces. In particular we address the power-distance metric and the additive-weight metric. Power-distance and additive-weight metric Voronoi diagrams are well known and have several applications [5, 6, 7, 8, 20, 22, 41, 43]. Below we give a brief introduction to the properties of Voronoi diagrams in these metric spaces which are important to the results appearing in this thesis.

The Power-Distance Metric

In the *power-distance metric*, a weight $w_p \geq 0$ is provided with each point $p \in P$. The distance between p and q is defined to be $pow(p,q) = d(p,q)^2 - w_p^2$ where $d(p,q)$ is the euclidean distance between p and q. The following geometric interpretation can be given to $pow(p,q)$: Let L be the line through q that is tangent to a disk with radius w_p^2 and center p. Then $\sqrt{pow(p,q)}$ is the length of the segment between q and the point on L tangent to the disk. Given two points $p, q \in P$, the locus of points with equal weighted distance from both p and q is a straight line. Hence, the concept of k^{th}-order Voronoi diagram generalizes nicely to this metric. Aurenhammer [5, 6] and Edelsbrunner [27] discuss some properties of k^{th}-order Voronoi diagrams in the power-distance metric, and they call these diagrams k^{th}-order power diagrams. The properties that are most important to us include the facts that the edges defining the k^{th}-order power diagram are straight lines and that the diagram contains only $O(k^2 n)$ faces. We discuss these diagrams in more detail in Section 8.2.

Weighted Metrics

In the *additive-weight metric*, the distance between p and q is defined to be $d(p,q)+w_p$. The k^{th}-order Voronoi diagram for this metric (called a *weighted Voronoi diagram*) has only $O(kn)$ edges. Rosenberger [41] and Ash and Bolker [4] have shown that such diagrams can be computed in $O(k^2 n \log n)$ time using $O(kn)$ space. The edges bounding the regions of k^{th}-order weighted Voronoi diagrams are second degree curves. [41] As it turns out, we are able to apply our compaction techniques to the k-nearest neighbors problem in the additive-weight metric. This serves as an example to illustrate the fact that the techniques presented here, and fully developed for the euclidean metric, can be extended to other metric spaces.

7.2 Planar Point Location

In the construction of our data structure for query-retrieval problems, we use a planar point location data structure due to Kirkpatrick. [35] The planar point location problem assumes a coordinate embedding of a planar graph (such as a Voronoi diagram) in R^2. If the graph has n edges then we can build a data structure which can answer the following query in $O(\log n)$ time: *which face of the planar graph does the query point lie in ?* Kirkpatrick's planar point location data structure requires only $O(n)$ space and can be constructed with $O(n \log n)$ preprocessing time.

A different planar point location technique is given by Edelsbrunner, Guibas, and Stolfi [28]. This technique also achieves linear space and logarithmic time planar point location. It has the advantage over Kirkpatrick's technique in that it can be generalized to to solve point location problems in sub-divisions of the plane that are defined by curved edges such as hyperbolas. The fact that we are able to do optimal time and space planar point locations in sub-divisions defined by curves is important in latter sections of this thesis where we extend our compaction techniques to Voronoi diagrams in non-euclidean metric spaces.

7.3 The Voronoi Diagram Compaction Technique

At this point, we are ready to present the main results of Part II of this thesis. Let V be the *triangulated* k^{th}-order Voronoi diagram for a set P of n points in the plane under the standard L_2 (euclidean) metric. We assume that the points of P are in general position. Then V is a planar graph with $O(kn)$ edges and $O(kn)$ faces [36]. Using Kirkpatrick's results [35] on planar point location, it is well known how to preprocess V to answer k-nearest neighbor queries in $O(\log n+k)$ time. Unfortunately, the resulting data structure occupies $O(kn)$ space. In this section we construct an optimal data structure using only $O(n)$ space which achieves the optimal $O(\log n + k)$ time bound for answering k-nearest neighbor queries.

7.4 Planar Separator Techniques

We begin by taking the dual of V: $\tilde{V}$. Since the points are in general position, $\tilde{V}$ is a planar, degree 3 graph with $O(kn)$ edges and nodes. A node v of $\tilde{V}$ corresponds to a region of the plane. Any query point in this region has the same set S_v of k-nearest neighbors in P. It follows from Observation 7.1.1 that whenever v and w are adjacent nodes in $\tilde{V}$, S_v and S_w differ in at most one point. For each edge $e = (v, w)$ in $\tilde{V}$, label e with the set of points $L_e = S_v \bigcap S_w$. So L_e contains either $k-1$ or k points. Now fix a point $p \in P$ and consider the faces of V which contain p in their list of k-nearest neighbors. These faces correspond to a set of nodes in $\tilde{V}$. Let $(\tilde{V})_p$ consist of these nodes, together with the edges e such that $p \in L_e$. Then we have the following observation:

Observation 7.4.1 *For all $p \in P$, $(\tilde{V})_p$ is connected.*

Proof: Consider a node $v \in (\tilde{V})_p$ and the corresponding face F_v in V. Draw a straight line from any point in F_v to p. Moving along this line, we are always getting closer to p, so each face of V which we pass through contains p as one of its k-nearest neighbors. Furthermore, since they are adjacent, these faces which the line passes through correspond to a connected subset of nodes in $(\tilde{V})_p$. Let w be the node in $(\tilde{V})_p$ whose corresponding face in V contains p. Then we have shown that v is in the same connected component as w for all $v \in (\tilde{V})_p$. □

At this point, we would like to partition $\tilde{V}$ into groups containing $\sim k^6$ nodes. Each connected component of a group will then correspond to a contiguous group of faces in the triangulated Voronoi diagram. Using the Lipton-Tarjan planar separator theorem [39], we can split $\tilde{V}$ in half by removing $O(\sqrt{kn})$ edges. This can be accomplished in $O(kn)$ time. Continue to apply the separator theorem, at each level splitting the existing groups in half until the point is reached where groups contain $\sim k^6$ nodes. At this point $O(\frac{n}{k^2})$ edges will have been removed and we will have used $O(kn \log(\frac{n}{k^6}))$ time running the planar separator algorithm. Let $\tilde{V}_1, \ldots, \tilde{V}_l$ be

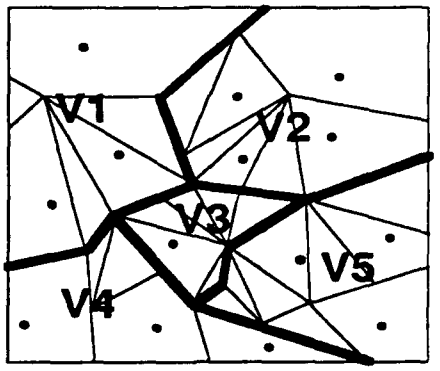

Figure 7-1: The triangulated Voronoi diagram V with darker edges defining $V_1, \ldots, V_l$

the resulting $l = O(\frac{n}{k^6})$ pieces of $\tilde{V}$. Then define P_i to be the union of the S_v such that $v \in \tilde{V}_i$.

Observation 7.4.2 *For any query point q, the k-nearest neighbors of q are completely contained in some P_i.*

Proof: Let F_q be the face in V which contains q and v_q be the corresponding node in the dual $\tilde{V}$. Then $v_q \in \tilde{V}_i$ for some i and the set of k-nearest neighbors to q is $S_{v_q} \subset P_i$. □

For all i, let V_i be the region of the plane defined by $\tilde{V}_i$. See Figure 7-1. Notice that collectively there are at most $O(\frac{n}{k^2})$ connected sub-regions among the regions $V_1, \ldots, V_l$. These connected sub-regions are defined by the $O(\frac{n}{k^2})$ edges of $\tilde{V}$ that are removed during the planar separator process and any V_i may be the union of numerous connected sub-regions. Using Kirkpatrick's planar point location algorithm [35], preprocess this set of edges to create a data structure which on input q, can locate the region V_i containing q. Call this the *locator tree* for $V_1, \ldots, V_l$. This tree has space linear in the number of edges needed to bound the V_i's. Furthermore, since Kirkpatrick's planar point location data structure on m regions can be computed in $O(m \log m)$ time, the locator tree can be constructed in $O(\frac{n}{k^2} \log n)$ preprocessing time.

Observation 7.4.3 *The locator tree uses only $O(\frac{n}{k^2})$ space.*

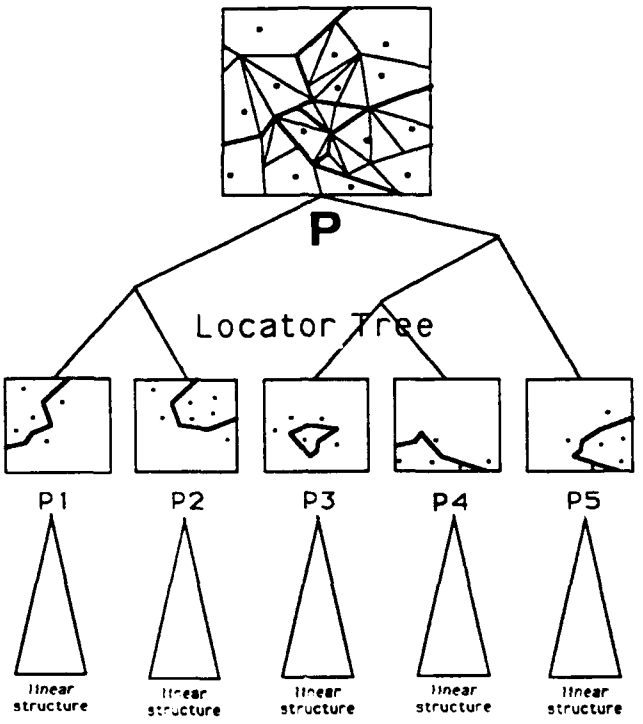

Figure 7-2: Linear structures for the P_i's give a linear structure for P

Proof: Each boundary edge of a region V_i corresponds to a removed edge in the dual. We have removed at most $O(\frac{n}{k^2})$ edges. □

Once we have located the V_i which contains q, we need only search P_i to find the k-nearest neighbors of q. Since P_i was defined from $\sim k^6$ sets of points S_v for $v \in \tilde{V}_i$, and each S_v contains k points, $\|P_i\| \leq k^7$. Hence by traversing the locator tree in $O(\log n)$ time, we can reduce the original query problem to one of finding the k-nearest neighbors out of a set P_i with only $O(k^7)$ points. At this point, one should notice that the sets $P_1, \ldots, P_l$ are **not** pairwise disjoint. In fact, we have $\sim \frac{n}{k^6}$ P_i's, each of size $O(k^7)$, and a naive analysis would suggest that $O(kn)$ memory is required to store the P_i's. In order to obtain an $O(n)$ upper bound on the size of the data structure we are building, it is necessary to give a strict upper bound on the number of duplications of points which occur among the P_i's. Toward this end, we show that the total number of points, counting duplications, which need to be stored in the data structure is bounded above by $(1 + O(\frac{1}{k}))n$:

Lemma 7.4.4 $\sum_{i=1}^{l} \|P_i\| \leq (1 + O(\frac{1}{k}))n$.

Proof: We begin by assigning each point $p \in P$ to the node $v_p \in \tilde{V}$ whose corresponding face in V contains p. Now we argue that the number of duplicates of points in P which are contained in the P_i's is bounded by k times the number of edges cut during the planar separator stage of the construction. Let $x \in P_i$. Then $x \in S_v$ for some $v \in \tilde{V}_i$. So $v \in (\tilde{V})_x$ which is connected by Observation 7.4.1. Then either $(\tilde{V})_x$ is completely contained in $\tilde{V}_i$ (and x occurs uniquely in P_i) or else an edge e of $(\tilde{V})_x$ was cut during the construction of $\tilde{V}_i$ (i.e., a node of $\tilde{V}_i$ must be incident to a cut edge e for which $x \in L_e$). Since L_e contains at most k points for any e and at most $O(\frac{n}{k^2})$ edges e have been cut, the total number of duplicate points in all of the P_i's is $O(\frac{n}{k})$. □

7.5 Probabilistic Techniques

Each of the sets of points P_i will now be treated separately. It follows from Lemma 7.4.4 that by creating a linear size data structure for k-nearest neighbor queries on each P_i, we can build a linear size data structure for k-nearest neighbor queries on P using the locator tree discussed above. See Figure 7-2 for a picture of such a data structure.

7.5.1 Review of Chernoff Bounds

Many of the results in this section use mathematical theorems for bounding the tail of a binomial distribution. These results are referred to loosely as *Chernoff bounds* and can be found in many sources [3, 21, 30, 42].

Let $X_1, \ldots, X_k$ be independent $0-1$ random variables with probability q of being 1. Then let $\chi = \sum_1^k X_i$, and $\beta \geq 1$. We would like to bound the probability that χ exceeds its expected value qk by more than a factor β. The following result is a well known Chernoff bound and can be easily derived using moment generating functions.

Lemma 7.5.1 $Pr[\chi \geq \beta qk] \leq exp((\beta - 1 - \beta \ln \beta)qk)$

7.5.2 Assigning the P_i to Buckets

For each P_i create a set of $s = \frac{k}{\log^2 k}$ buckets $B_i^1, \ldots, B_i^s$. We assign the points of P_i to these buckets with equal probability. That is, a point $p \in P_i$ is assigned to bucket B_i^j with probability $\frac{\log^2 k}{k}$. Our intention is to search each bucket for the $\sim \log^2 k$ nearest neighbors of q, and in this manner capture the k-nearest neighbors of q in P_i. To do this, we need to show that for any q, the k-nearest neighbors of q are evenly distributed among the buckets. Consider the k^{th}-order Voronoi diagram on P_i. This diagram has $m = O(k^8)$ regions corresponding to all possible sets of k-nearest neighbors to q. Call the corresponding sets of k points: $C_1, \ldots, C_m$ the *constraints.* We say that a constraint C_r is *satisfied* if each bucket $B_i^1, \ldots, B_i^s$ contains less than $\log^2 k + O(\log^{\frac{3}{2}} k)$ points from C_r.

Lemma 7.5.2 *With probability* $\geq 1 - \frac{1}{k}$ *every constraint is simultaneously satisfied by a random assignment of* P_i *to the buckets.*

Proof: This is a direct applications of the Chernoff bound given in Lemma 7.5.1: $\Pr[\chi \geq \beta qk] \leq exp((\beta - 1 - \beta \ln \beta)qk)$ Choose a constraint $C_r = \{p_1, \ldots p_k\}$ and for $l = 1 \ldots k$ let $X_l = 1$ if $p_l \in B_i^j$. In this case $q = \frac{\log^2 k}{k}$. We say that C_r is satisfied if less that $\log^2 k + \gamma \log^{\frac{3}{2}} k$ points from C_r are contained in any one bucket. We will show that γ can be chosen to satisfy the lemma. Now let $\beta = 1 + \frac{\gamma}{\sqrt{\log k}}$. Then C_r is satisfied if less than βqk points from C_r are contained in any one bucket. Since there are $O(k^8)$ constraints and $O(k)$ buckets, from the Chernoff bound given above, we conclude that the probability that some constraint is not satisfied is less than $k^9 \Pr[$ the number of points from C_r in bucket $B_i^j \geq \log^2 k + \gamma \log^{\frac{3}{2}} k] \leq k^9 exp((\beta - 1 - \beta \ln \beta)qk) \leq k^9 exp(-\frac{\gamma^2 \log k}{2} + \frac{\gamma^3 \sqrt{\log k}}{2})$. Hence, in order to satisfy the lemma, it suffices to pick γ such that $k^9 exp(-\frac{\gamma^2 \log k}{2} + \frac{\gamma^3 \sqrt{\log k}}{2}) \leq \frac{1}{k}$. When k is sufficiently large, $\gamma \geq 4$ is good

enough. □

For each set P_i, assign the points randomly to buckets and then check all of the constraints to see that they are satisfied. Lemma 7.5.2 tells us that with probability $\geq 1 - \frac{1}{k}$ the assignment to P_i will check out on the first try. We can build the k^{th}-order Voronoi diagram on P_i in $O(k^2\|P_i\| + \|P_i\| \log(\|P_i\|))$ time [1, 36]. By Lemma 7.5.2 we expect to check at most 2 assignments in order to successfully divide P_i into buckets. Hence the expected time to assign all of the P_i's to buckets is $O(k^2 n + n \log n)$. Here we have described a Las Vegas algorithm to divide up the regions P_i into buckets. The assignments to buckets are guaranteed to satisfy all constraints, but the running time of the algorithm is not guaranteed. However, with high probability this algorithm will terminate in $O(k^2 n)$ time. In the next section we show how to assign the P_i's to buckets deterministically.

7.6 The Compacted Data Structure

At this point, we have a data structure which divides P up into sets $P_1, \ldots, P_l$ with *very little duplication of points*. Given a query point q, we can locate the proper set P_i which contains q's k-nearest neighbors in $O(\log n)$ time. Each P_i is further divided up into $\frac{k}{\log^2 k}$ buckets. We have to retrieve the $\log^2 k + 4 \log^{\frac{3}{2}} k$ nearest neighbors to q from each bucket $B_i^1, \ldots, B_i^s$. The idea now is to recursively build a data structure for retrieving $\log^2 k + 4 \log^{\frac{3}{2}} k$ nearest neighbors in each bucket. At the bottom level, where we are searching buckets for the c (a fixed constant which does not vary with k) nearest neighbors of q, we simply construct c^{th}-order Voronoi diagrams on the points in these buckets and process these diagrams using Kirkpatrick's planar point location techniques to get linear space, $O(c)$ time structures for finding the c-nearest neighbors to q. The c points retrieved from each leaf are stored in a global linear array of possible neighbors for q. In what follows, we will show that at most $O(k)$ points are stored in this array. This array of $O(k)$ points is guaranteed to contain the k-nearest

neighbors of q. To find the k-nearest neighbors, we will begin by finding the k^{th}-nearest point, x, to q in this array in $O(k)$ time by using an order statistic algorithm [12]. Then simply compare each point to x and keep those which are closer or equal in distance to q. Notice that as the k-nearest neighbors algorithm traverses the data structure we have constructed at the intermediate levels no points are inserted into the array. Points are only retrieved and written to the array in the leaves, and points are not passed recursively up the data structure. Once the entire structure has been traversed, and all of the relevant leaves have been investigated, then the k-nearest neighbors to the query point are eliminated from among those present in the global array.

Below we give a careful analysis of the data structure to show that it occupies $\Theta(n)$ space. We also solve the necessary recurrence to prove that the structure reports a set of $O(k)$ points containing the k-nearest neighbors of a query point q in $O(\log n + k)$ time.

The data structure we have constructed is a tree with levels alternating between locator trees and pointers to buckets. Notice that at level i of this structure we are breaking up a search problem for k_i neighbors into a number of new search problems for $k_{i+1} = \log^2 k_i + O(\log^{\frac{3}{2}} k_i)$ neighbors. Hence at level i, $\Theta(\overbrace{\log\log\ldots\log}^{i} k)^2$ is a lower bound on the range of the neighbor searches being performed. Now notice from Observation 7.4.3 that the locator trees at level i will occupy $O(\sum_j \frac{m_j}{k_{i-1}^2})$ space, where m_j are the sizes of the buckets on level $i-1$. By repeated applications of Lemma 7.4.4: $\sum_j m_j \leq (1+\frac{1}{k})\cdots(1+\frac{1}{k_{i-1}})n$. Hence, the total amount of space occupied by the locator trees is: $O(\frac{n}{k^2} + \frac{(1+\frac{1}{k})n}{(\log k)^2} + \frac{(1+\frac{1}{k})(1+\frac{1}{\log^2 k})n}{(\log\log k)^2} + \cdots) = O(n)$ space. Since the number of bucket nodes increases geometrically every other level, the space used to store the bucket nodes is dominated by the space used to store the leaves of the data structure. Each leaf holds a constant sized Voronoi diagram. Hence the total amount of space used to store this structure is bounded by a constant times the total number of points (including duplicates) which appear in the Voronoi diagrams in the leaves.

By Lemma 7.4.4 this number is $O(((1+\frac{1}{k})(1+\frac{1}{(\log k)^2})(1+\frac{1}{(\log\log k)^2})\cdots)n) = O(n)$.

The upper bound on running time for extracting the k-nearest neighbors is equal to the time required to traverse the first level locator tree, plus upper bounds on the times required to search recursively through the first level buckets:

$$\begin{aligned} T(n,k) &\leq \alpha \log n + \frac{k}{\log^2 k} T(k^7, \log^2 k + 4\log^{\frac{3}{2}} k) \\ T(n,c) &\leq \alpha c \end{aligned}$$

This can be solved by substitution using the following expression for $T(n,k)$:

$$\alpha \log n + \frac{6\alpha k}{\log k} + \frac{6\alpha(1+\frac{4}{\sqrt{\log k}})k}{\log\log k} + \frac{6\alpha(1+\frac{4}{\sqrt{\log k}})(1+\frac{4}{\sqrt{\log\log k}})k}{\log\log\log k} + \cdots$$

Hence, we have $T(n,k) = O(\log n + k)$.In the case that the query point q is restricted to a set of $O(n)$ points, then this time can be reduced to $O(k)$. To see this. let $c_1, \ldots, c_m$ be the possible values for q where $m = O(n)$. Eliminate the locator tree at the first level of the compacted data structure and replace it with an array $A[1 \ldots m]$ where $A[j]$ contains a pointer to the node on the second level which handles the set P_i containing the k-nearest neighbors of c_j. On input c_j, begin traversing the compacted data structure at the node indicated by the pointer $A[j]$. In the running time analysis this replaces the $\alpha \log n$ term with $O(1)$, and the recurrence for $T(n,k)$ now has solution $O(k)$.

Lastly, we must check that at most $O(k)$ points are reported. Notice that points are reported only in the leaves of this structure and are not *passed up* the tree and filtered at each level. In fact, if we computed order statistics and filtered points at each level, then the resulting report time would have been $\Omega(\log n + k\log^* k)$. Implicit in the recurrence given above is the fact that the superset of the k-nearest neighbors collectively reported by the leaves has size at most: $((1+\frac{4}{\sqrt{\log k}})(1+\frac{4}{\sqrt{\log\log k}})(1+\frac{4}{\sqrt{\log\log\log k}})\cdots)k = O(k)$. To see this notice that at the first level we are neighbor

searching for $\log^2 k + 4\log^{\frac{3}{2}} k$ points from each of $\frac{k}{\log^2 k}$ buckets. So at the first level, the number of points which are going to be reported is bounded above by $(1+\frac{4}{\sqrt{\log k}})k$. A similar argument adds an additional factor of $(1+\frac{4}{\sqrt{\log\log k}})$ at the second level, etc.

7.7 Extensions to Voronoi Diagrams in Other Metric Spaces

The ideas used above to construct a linear-sized data structure for optimal-time k-nearest neighbor searching are general enough to apply to Voronoi diagrams constructed using a wide variety of distance functions. In fact, it is clear from our discussion that the techniques work whenever the k^{th}-order Voronoi diagram is planar and contains $O(k^\beta n)$ edges and faces for some constant β. The only problem may be that the edges of a Voronoi diagram in an alternative metric may not be straight lines. In that case, a straightforward application of Kirkpatrick's planar point location algorithm will not be sufficient for the construction of the locator trees. However, in many cases involving edges defined by curves, we may be able to use a planar point location algorithm given by Edelsbrunner, Guibas, and Stolfi [28] and described in Section 7.2. Given that the diagram has straight edges or that curved edges can be handled, we can always apply the planar separator theorem to build a locator tree leading to sets P_i containing $O(k^{\beta+6})$ points. At this point, we will have $O(k^{2\beta+6})$ constraints imposed on the buckets for P_i. As in the L_2 case, these constraints can be satisfied with high probability. The resulting data structure obtained by applying these techniques recursively satisfies the same recurrences as above with slightly different constants. Applying these ideas to non-euclidean Voronoi diagrams (such as those generated by the power-distance metric or weighted metric with additive weights) is important in the applications presented in Chapter 8. Note the the Voronoi diagrams for these metrics have $k^{O(1)}$ edges and the connectivity property described in Observation 7.4.1.

7.8 Removing Randomness from the Construction

To make the above construction deterministic, we use the well known *probabilistic method* described in [42]. For each P_i, we will proceed by first assigning the points to two buckets B_0 and B_1. Then each of these buckets will be split into two buckets, and then into four buckets, etc., until after $\log k - 2\log\log k$ iterations the points of P_i have been partitioned into $\frac{k}{\log^2 k}$ buckets. During the initial iteration, the points of P_i are taken in some order $p_1, p_2, \ldots$ and assigned to B_0 or B_1. The probabilistic method allows us to ensure that each bucket ends up containing $\leq \frac{k}{2} + O(\sqrt{k\log k})$ points from each constraint C_j. Central to the methods in [42] for solving this problem is the computation of the conditional probabilities: Pr[$\geq \frac{k}{2} + 4\sqrt{k\log k}$ points from C_j are assigned to $B_0 \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0$]. Call this conditional probability Pr[$C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0$]. Such a conditional probability can be computed exactly as a binomial series with $\leq k$ terms. We will come back to the problem of computing the Pr[$C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0$]'s shortly. First, however, we detail the probabilistic method used to assign the points.

Suppose that points $p_1, \ldots p_m$ have been assigned to buckets B_0 and B_1 in such a manner that there still exists an assignment of the remaining points which does not violate any of the constraints. We would now like to assign p_{m+1} to one of the buckets in a manner which preserves the property that an assignment of the remaining points exists which does not violate any of the constraints. So for each of the constraints, C_j, effective in set P_i, we compute the conditional probability for $B = B_0, B_1$: Pr[$C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0, p_{m+1} \in B$].

By hypothesis, for either $B = B_0$ or $B = B_1$ (or perhaps both) the sum of these conditional probabilities will be less than 1: $\sum_j$ Pr[$C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0$ $p_{m+1} \in B$] < 1. To see this, notice that we have assumed as our induction hypothesis that $\sum_j$ Pr[$C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0$] < 1. But Pr[$C_j \mid$

$p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0] = \frac{1}{2}\Pr[\ C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0\ p_{m+1} \in B_0]$ $+ \frac{1}{2}\Pr[\ C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0\ p_{m+1} \in B_1]$. Hence, $\frac{1}{2}\sum_j \Pr[\ C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0\ p_{m+1} \in B_0] + \frac{1}{2}\sum_j \Pr[\ C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0\ p_{m+1} \in B_1] < 1$. It follows that one of the two sums on the left hand side of this inequality must be less than 1. In the base case, where none of the points have been assigned to any buckets yet, Lemma 7.5.2 assures us that the initial sum will be less than 1. We then assign p_{m+1} to a bucket which achieves a favorable set of conditional probabilities. In the course of this process, we have guaranteed that each bucket gets split roughly in half for $\log k - \log\log k$ levels. By construction, the maximum number of points from any constraint C_j in a given bucket at stage i obeys the following recurrence: $m_i \leq \frac{m_{i-1}}{2} + O(\sqrt{m_{i-1}\log k})$ where $m_0 = k$. It follows by substitution that $m_i \leq \frac{k}{2^i} + O(\sqrt{\frac{k}{2^i}\log k})$ for all i such that $\frac{k}{2^i} = \Omega(\log k)$. Hence, at the stage numbered $\log(\frac{k}{\log^2 k})$, we have produced roughly equal sized buckets $P_1, \ldots P_{\frac{k}{\log^2 k}}$ such that each bucket contains at most $\log^2 k + O(\log k\sqrt{\log k})$ points from any C_j.

We now turn our attention to computing the conditional probabilities. Let u be the number of points from C_j which have not been assigned to buckets yet. Let $v = (\frac{k}{2} + 4\sqrt{k\log k})$ - #(points from C_j that have been assigned to B_0). Define $c(u,v)$ $= \frac{1}{2^u}\sum_{l=v}^{u}\binom{u}{l}$ Then $\Pr[\ C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0\ ] = c(u,v)$.

To compute these conditional probabilities efficiently, it will suffice to precompute all $\binom{u}{l}$ for $u \leq k$ and $l \leq k$. Since $\binom{u+1}{l} = \frac{u+1}{u-l+1}\binom{u}{l}$ and $\binom{u}{l+1} = \frac{u-l}{l+1}\binom{u}{l}$, this precomputation can be achieved iteratively in $O(k^2)$ time. Iteratively precompute all of the $c(u,v)$ for all $u \leq k$, $\frac{k}{2} \leq v \leq k$. We can then simply look up the conditional probabilities, and each $\Pr[\ C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0\ ]$ requires $O(1)$ time to lookup. The number of constraints active in P_i is bounded above by the number of faces in the k^{th}-order Voronoi diagram on P_i: $O(k\|P_i\|)$. Hence, there are $O(k\|P_i\|)$ active constraints, C_j, in P_i. After precomputing all of the $c(u,v)$, $\sum_j \Pr[$

$C_j \mid p_1 \in B_0, p_2 \in B_1, \ldots, p_m \in B_0]$ requires $O(k\|P_i\|)$ to compute. This computation is performed $O(\log k)$ times to assign all points to buckets. Hence, the total time required to assign points to buckets is $O(k \log k\|P_i\| + k^2) = O(k \log k\|P_i\|)$. Hence, the entire assignment process for the first level of the structure takes $O(k \log kn)$ time. At subsequent levels, we are building structures for the k_i-nearest neighbor problem, where $k_i = \log^2 k_i + O(\log^{\frac{3}{2}} k_i)$ Hence, the bound $O(k_i \log k_i n)$ for assigning points to buckets at each level decreases more than geometrically fast.

7.9 Total Preprocessing Time

The total computation time for building the compressed Voronoi diagram is given by the sum of the times for the following operations. The time given in each case is the sum of the associated operations performed at every level:

1. compute all intermediary Voronoi diagrams: $O(k^2 n + n \log n)$ [1, 36].

2. compute planar separators to build P_i's: $(kn \log \frac{n}{k^6})$ [39].

3. build all of the locator trees: $O(n \log n)$ [35].

4. A deterministic assignment of the P_i's to buckets: $O(k^2 n \log k)$ [Section 7.8].

5. A randomized assignment of the P_i's to buckets: $O(k^2 n + n \log n)$ expected time [Section 7.5].

At the i^{th} level of the construction, we are dealing with k_i-nearest neighbor problems where $k_1 = k$ and $k_{i+1} = \log^2 k + \log^{\frac{3}{2}} k_i$. (For convenience we designate $k_0 = n$.) Since the k^{th} order Voronoi diagram can be computed in $O(k^2 n + n \log n)$ time, at the i^{th} level computing the Voronoi diagrams for all the P_js costs $\sum_{P_j} O(k_i^2\|P_j\| + \|P_j\| \log \|P_j\|) = \sum_{P_j} O(k_i^2\|P_j\| + \|P_j\| \log k_{i-1}) = O(k_i^2 n + n \log k_{i-1})$. Summing over all levels, we see that the total time spent computing Voronoi diagrams

is $O(k^2n + n\log n)$. Likewise as we showed in Section 7.4, at level i the planar separators computed to break up P_j require $O(k_i\|P_j\|\log(\frac{\|P_j\|}{k_i^6}))$ time to compute. Summing over all i gives the bound stated above. Finally, we have shown above that the locator trees at level i take time $O(\frac{n}{k_i^2}\log n)$ time to construct. Summing over all i gives a total preprocessing time of $O(n\log n)$ for building locator trees.

The dominating term in the sum of these time bounds varies with k, but it is clear that the construction can always be accomplished in $O(k^2n\log k + kn\log n)$ deterministic time or $O(k^2n + kn\log n)$ randomized time.

7.10 A Monte Carlo Construction

Using a monte carlo construction, we can build a compacted data structure with $O(n\log^2 n\log\log n)$ preprocessing time. For $k \leq \log n$, the deterministic construction has preprocessing time $O(n\log^2 n)$. For $k \geq \log n$, the preprocessing time for the above deterministic construction is dominated by the computation of Voronoi diagrams and the assignment of points to buckets *at the one or two top levels.* However, when $k \geq \log n$, we can discard the initial computation of Voronoi diagrams, and assign points directly to buckets in a randomized manner. In what follows we show that with high probability this approach builds a correct data structure for solving the k-nearest neighbors problem in $O(n\log^2 n\log\log n)$ preprocessing time. Furthermore, each time we query this data structure we get a *witness* which tells us whether or not the answer provided to our query is correct.

When $k \geq \log n$, skip the initial planar separator stage, and at the first level assign the points directly to $\frac{k}{\log n}$ buckets using $O(n)$ time. As we demonstrated in the previous analysis of probabilistic bucket assignment, with high probability, in order to capture the k-nearest neighbors of q it suffices to report the $O(\log n)$ nearest neighbors of q in each bucket. This can be accomplished using the above deterministically constructed data structures requiring $O(n\log^2 n\log\log n)$ total preprocessing

time. Hence the total preprocessing time to build this Monte Carlo structure is $O(n \log^2 n \log\log n)$. If we use a randomized assignment of points with checking (as presented in Section 7.5) at the first level, this time bound then reduces to $O(n \log^2 n)$.

Given a query q to this new data structure, keep track of the $O(\log n)$ points reported from each of the first level buckets. We are certain that each bucket reported correctly. Now let $p_1, \ldots, p_k$ be the final report from this new structure. If our top level assignment of points to buckets is correct, there should be at least one point reported from each bucket that does not make the final list $p_1, \ldots, p_k$. These $\frac{k}{\log n}$ discarded points are our witnesses that the final list is correct. If all of the points reported from some bucket are contained in the final list, then that bucket contains more than $\log^2 k + 4\log^{\frac{3}{2}} k$ of the nearest neighbors to q and the top level assignment of points to buckets violates at least one constraint.

Chapter 8

Applications of the Compaction Technique

8.1 k-Nearest Neighbor and Circular Range Search

Given a set of n points in the Euclidean plane E^2, and a fixed k, the technique given in Chapter 7 provides an $O(n)$ space data structure for computing the k-nearest neighbors of a query point q. However, if k is not fixed, but rather given as a part of the query, then we simply construct $\log n - \log\log n$ linear size data structures $D_1, \ldots, D_l$, where the D_i can be used to obtain the $2^i \log n$-nearest neighbors of q. These $O(\log n)$ structures occupy a total of $O(n \log n)$ space. Now, given a pair (q, k), we simply compute the smallest j such that $2^j \geq \lfloor \frac{k}{\log n} \rfloor$ and use D_j to compute the $2^j \log n$-nearest neighbors of q in $O(\log n + 2^j \log n) = O(\log n + k)$ time. From this set of $O(k + \log n)$ neighbors, filter out the k-nearest in $O(k + \log n)$ additional time by finding the k^{th} order statistic x [12] and keeping the points which are as close to q as x.

To solve the circular range search problem we also use the $\log n - \log\log n$ data

structures described above. When given a query disk with center c and radius r, use q to query $D_1, D_2, \ldots$ until some structure D_j provides output which contains a point x farther than distance r from q. This is a straightforward application of the filtering search technique introduced by Chazelle [13].

The central idea in filtering search is that the *search* and *report* parts of a query-retrieval algorithm should be made dependent upon each other. In this manner, the more points the algorithm is going to report, the longer it is allowed for searching. We can see that considering $D_1, D_2, \ldots$ in order is an effective means of balancing searching and reporting in the circular range search problem. If, after searching D_i, we find that all points reported are still closer than distance r from point q, we then know that the report part of the algorithm will run for at least $\Omega(2^i)$ time (to report $\Omega(2^i)$ points) and hence we have enough time to search D_{i+1} without degrading the asymptotic running time of the algorithm.

Now, let j be the smallest index such that D_j outputs a point x outside the query disk. Then the time to execute these j stages is simply: $O(\sum_{i=0}^{j}(2^i + 1)\log n) = O(2^j \log n + \log n) = O(k + \log n)$ since $k = O(2^j \log n)$. Finally, search through the $2^j \log n$ points given by D_j and output those which are contained inside the query disk.

8.2 Half Space Range Search and Power Diagrams

Recall that for the half-space range search problem in 3 dimensions, we are given a set P of n points in E^3 and a query half-space. We are asked to report all points of P that lie inside (or on the boundary of) this half-space. In this section, we show how this problem can be transformed into a nearest neighbor search problem in the power-distance metric. To begin, we show how to transform the range search problem into a ray-stabbing problem using geometric duality [19, 29].

Geometric Duality

In the half-space range searching problem, we are given a set of points in 3-space (a_i, b_i, c_i) for $i = 1 \ldots n$. Then for each query half-space $\alpha x + \beta y + \gamma z \geq 1$ we are asked to return those points which are contained in the half-space or on its boundary. Geometric duality defines that each point (a_i, b_i, c_i) has a dual half-space given by the equation $a_i x + b_i y + c_i z \geq 1$. Similarly, the query half-space has a dual point (α, β, γ). Then we have the following duality result:

Observation 8.2.1 *A point (a_i, b_i, c_i) lies inside the half-space $\alpha x + \beta y + \gamma z \geq 1$ iff the dual point (α, β, γ) lies inside the dual half-space $a_i x + b_i y + c_i z \geq 1$.*

Proof: Both sides of the "iff" are equivalent to: $\alpha a_i + \beta b_i + \gamma c_i \geq 1$. □

Such simple duality results have powerful consequence in computational geometry. [19] For our purposes, Observation 8.2.1 indicates that the transformation from a half-space range searching problem to a ray stabbing problem can be accomplished in $O(n)$ time and space. After the transformation, we are given a set of half-spaces defined by $a_i x + b_i y + c_i z \geq 1$ for $i = 1 \ldots n$ and a query point (α, β, γ). Our algorithm should report those half-spaces which contain this query-point.

Without loss of generality, rotate the coordinate axes so that none of the planes defining our half-spaces is parallel to the z-axis. Each half-space now intersects the z-axis. Define those half-spaces which contain a segment of the z-axis heading for $+\infty$ to be *upwardly oriented.* Define the others to be *downwardly oriented.* We will split these groups into two sets and treat them separately. This will cause us to run the algorithm twice, once on the upwardly oriented half-spaces and once on the downwardly oriented half spaces. Within a constant factor, however, this will not affect our running time. So, without loss of generality, we will assume that all of the half-spaces are upwardly oriented. Hence, if we consider a ray which originates at point (α, β, γ) and extends downward to $z = -\infty$ running parallel to the z-axis. the point (α, β, γ) is contained in exactly those half-spaces whose boundary planes

the ray intersects. In this manner, we have transformed the half-space range query problem into a vertical ray-stabbing problem.

In order to describe how to solve the ray-stabbing problem using k^{th}-order power diagrams we need to understand more about the relationship between power diagrams and planes in euclidean 3-space.

Power Diagrams and Planes in 3-Space

Aurenhammer and Edelsbrunner [6, 5, 27] provide a correspondence between the arrangements of planes in 3 dimensions and power diagrams in 2 dimensions. This correspondence hinges on the notion of a k-set. Given a set of n planes in E^3, the *k-set* is the locus of all points in E^3 that satisfy the following properties:

1. Each point in the k-set belongs to one of the n planes.

2. For every point in the k-set there are exactly $n - k$ planes passing above it in the vertical dimension (i.e., the positive z-direction).

Two points are said to be in the same *face* of a k-set if they lie in the same plane and have exactly the same planes passing above them.

In the above definition of a k-set, we say that a plane lies above a point (a, b, c), if for some positive λ, the point $(a, b, c + \lambda)$ belongs to the plane. Notice that the boundary of a k-set is formed by the lines defining the intersection of pairs of planes. Let Δ be a face in a k-set. Then we define the projection of Δ to be the image of Δ projected onto a plane at $z = -\infty$: $\{(x, y)$: $(x, y, z) \in \Delta$ for some $z\}$. Now fix some k_0. It is not hard to see that the projection of all the faces of the k_0-set gives a division of the plane into convex sub-regions.

We now wish to establish a correspondence between the weighted points which define a power diagram, and planes in 3-space. For proofs and more detailed explanations of the results appearing below, we refer the reader to the work of Aurenhammer and Edelsbrunner [6, 5, 26, 27]. Recall that each point p_i in the set of

points defining a power diagram has an associated weight: $w(p_i)$. Define a plane $\pi(p_i)$: $z = 2(x,y)^T p_i - p_i^T p_i + w(p_i)$ where $(x,y)^T p_i$ is the dot product of (x,y) and (x_i,y_i) and $p_i = (x_i,y_i)$. Furthermore, for any two points p_i, p_j we define $\text{chor}(p_i,p_j)$ to be the locus of points equidistant from p_i and p_j in the power distance metric with weights $w(p_i)$ and $w(p_j)$. It is not hard to see that $\text{chor}(p_i,p_j)$ is a straight line. Furthermore, we have the following observation:

Observation 8.2.2 *The projection of $\pi(p_i) \bigcap \pi(p_j)$ onto the plane is $chor(p_i,p_j)$.*

In fact, a much stronger result can be proven. Consider the k-sets formed by the set of planes $\Gamma = \pi(p_1), \pi(p_2), \ldots \pi(p_n)$. Then for a fixed k we have:

Theorem 8.2.3 *The projection of the faces of the k-set of Γ onto the plane at $z = -\infty$ is the k^{th}-order power diagram on the points $p_1, \ldots, p_n$ with weights $w(p_i)$. Furthermore, if $\pi(r_1)$ is the plane containing one of the faces Δ of the k-set, and $\pi(r_2), \ldots, \pi(r_k)$ are the planes below Δ, then the k-nearest neighbors to the points in the projection of Δ are $r_1, \ldots, r_k$.*

Theorem 8.2.3 can be used to give an upper bound on the size of the k^{th}-order power diagram. Since the maximum number of faces in a k-set in 3 dimensions has been shown to be $O(k^2 n)$ (see Clarkson and Shor [24]), the number of edges in a k^{th}-order power diagram is $O(k^2 n)$. Consequently, if k is known ahead of time, then our compaction techniques can be used to obtain an $O(n)$ data structure which can be used to compute the k-nearest neighbors in the power metric in $O(k + \log n)$ time. On the other hand, if k is not fixed, but given as a part of the query, then we can construct $\log n - \log\log n$ linear data structures as in the previous section and obtain the k-nearest neighbors of q in $O(\log n + k)$ time. Again, this structure requires $O(n \log n)$ space.

Returning to the vertical ray-stabbing problem, $p_1, \ldots, p_n$ and $w(p_1), \ldots, w(p_n)$ are the points and weights corresponding to the power diagram generated by the projection of Γ. We are given a query point (α, β, γ), and are asked to determine the

planes that this point lies above. Let $D_1, D_2, \ldots, D_{\log n - \log\log n}$ be the compacted data structures described above. First, we query D_1 with the point (α, β) to determine the $\log n$-nearest points $r_1, \ldots, r_{\log n}$ in the power-distance metric. This takes $O(\log n)$ time. Then $\pi(r_1), \ldots, \pi(r_{\log n})$ are the lowest $\log n$ planes which might appear below (α, β, γ). We can then check each of these planes $\pi(r_i)$ in constant time to see if (α, β, γ) appears above $\pi(r_i)$. At this point, we have still used $O(\log n)$ time. If all of $\pi(r_1), \ldots, \pi(r_{\log n})$ appear below (α, β, γ), then continue on to check D_2 and retrieve the $2\log n$ nearest neighbors to (α, β). Proceed in this manner, using the filtering search techniques presented in the previous section. At some D_i, we will find the first plane $\pi(r_j)$ which lies above (α, β, γ). At this point, stop and report all of the planes from the set $\pi(r_1), \ldots, \pi(r_{\log n 2^i})$ which lie below (α, β, γ). Let us say that k planes are reported. Then the set of planes retrieved from D_i has size at most $2k$. Since at a minimum D_i gives us $\log n$ planes to search through, it is clear that this algorithm runs in $O(\log n + k)$ time.

8.3 k-Nearest Neighbors in the Weighted Metric

Finally, k-nearest neighbor searching in the weighted metric (with additive weights) can also be performed using data structures built from the compaction techniques given in this paper. As described in Section 7.1.2, the k^{th}-order Voronoi diagram for this metric has only $O(kn)$ edges. However, since these edges are defined by second degree curves we cannot use Kirkpatrick's planar point location algorithm to build the locator trees in the compact data structure. In this case, we use the planar point location data structure given by Edelsbrunner, Guibas, and Stolfi [28] and described in Section 7.2. With this change, the resulting compact Voronoi diagrams in the additive-weight metric give the same optimal time and space complexity performance for finding k-nearest neighbors as the euclidean structures. When k is fixed, the data structure occupies $O(n)$ space and responds to queries in $O(\log n + k)$ time.

8.3.1 Further Extensions of the Compaction Technique

The technique given in Section 7 for the k-nearest neighbors problem may have applications to other retrieval problems. For example, this technique can be modified to obtain an $O(n)$ space data structure that can be used to solve a retrieval problem in $\Theta(k + \log n)$ time for any class of objects with the following properties:

1. Each object in the class is a bounded or unbounded region of the plane and its boundary is some Jordan curve.

2. If we consider any n objects in this class then the number of disjoint *k-regions* is $O(k^\beta n)$ (for some constant $\beta > 0$). We define a k-region as a connected region of the plane such that any point contained in that region belongs to exactly k objects. Furthermore, each k-region must be adjacent to at most $O(k^\alpha)$ other regions for some constant $\alpha > 0$.

3. The partitioning of the plane into k-regions by the set of Jordan curves J defining the n objects must satisfy certain point location properties: given any sub-partitioning of the plane by some $O(m)$ sized subset of J, there exists a data structure of $O(m)$ size which can perform planar point location in $O(\log m)$ time.

It is not hard to see that most geometric objects (circles, ellipses, polygonal curves with at most a constant number of edges) satisfy these properties. Consequently, for most of these problems, given a fixed value of k, we can obtain an $O(n)$ data structure that reports the $\Theta(k)$ objects containing a query point q in $O(\log n + k)$ time. (If more or less than $\Theta(k)$ objects contain q then the structure would report the empty set in $O(\log n)$ time.) However, for all such cases, these results are not new since the techniques given in [16, 17] can be used to obtain simpler $O(n)$ space data structures. In a similar vein, we can use this technique together with filtering search to solve the 3-dimensional dominance reporting problem in $O(n \log n)$ space and $O(\log n + k)$ time, but again a simpler data structure that achieves the same space and time bounds has been given by Gabow et al. [31].

Bibliography

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P.W. Shor. A linear time algorithm for computing the voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, pages 591–604, 1989.

[2] A. Aggarwal, M. D. Hansen, and T. Leighton. Solving query-retrieval problems by compacting voronoi diagrams. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990. to appear.

[3] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, April 1979.

[4] P. F. Ash and E. D. Bolker. Generalized dirichlet tesselations. *Geometriae Dedicata*, 20:209–243, 1986.

[5] F. Aurenhammer. Power diagrams: properties, algorithms, and applications. *SIAM Journal on Computing*, 16:78–96, 1987.

[6] F. Aurenhammer. Voronoi diagrams - a survey. Technical report, Institutes for Information Processing, Graz Technical University, Austria, 1988.

[7] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17:251–257, 1984.

[8] F. Aurenhammer and H. Imai. Geometric relations among voronoi diagrams. *Geometriae Dedicata*, 27:65–75, 1988.

[9] V. E. Benes. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, July 1964.

[10] J. L. Bentley and H.A. Maurer. A note on euclidean near neighbor searching in the plane. *Information Processing Letters*, 8:133–136, 1979.

[11] M. W. Bern, H.J. Karloff, P. Raghavan, and B. Schieber. Fast geometric approximation techniques and geometric embedding problems. Manuscript in preparation.

[12] M. Blum, R. W. Floyd, V. Pratt, R. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, August 1973.

[13] B. M. Chazelle. Filtering search: a new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, August 1986.

[14] B. M. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

[15] B. M. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap. New upper bounds for neighbor searching. *Information and Control*, 68:105–124, 1986.

[16] B. M. Chazelle and H. Edelsbrunner. Optimal solutions for a class of point retrieval problems. *Journal of Symbolic Computation*, 1:47–56, 1985.

[17] B. M. Chazelle and H. Edelsbrunner. An improved algorithm for constructing k^{th}-order voronoi diagrams. *IEEE Transactions on Computers*, 36:1349 - 1354, 1987.

[18] B. M. Chazelle and H. Edelsbrunner. Linear space data structures for two types of range search. *Discrete and Computational Geometry*, 2:113–126, 1987.

[19] B. M. Chazelle, L.J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[20] B. M. Chazelle and F. P. Preparata. Half-space range search: an algorithmic application of k-sets. *Discrete and Computational Geometry*, 1(1):83–94, 1986.

[21] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23, 1952.

[22] L. P. Chew and R. L. Drysdale. Voronoi diagrams based on convex distance functions. In *Proceedings of the 1st Annual ACM Symposium on Computational Geometry*, pages 235–244, 1985.

[23] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Carnegie-Mellon Graduate School of Industrial Administration, Pittsburgh, PA, 1976.

[24] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 4(5):387–422, 1989.

[25] R. Cole and C.K. Yap. Geometric retrieval problems. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 112–121. IEEE. November 1983.

[26] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer–Verlag, Berlin, West Germany, 1986.

[27] H. Edelsbrunner. Edge-skeletons in arrangements with applications. *Algorithmica*, 1:93–109, 1986.

[28] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

[29] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[30] P. Erdos and J. Spencer. *Probabilistic Methods in Combinatorics.* Academic Press, New York, 1974.

[31] H. N. Gabow, J. L. Bentley, and R.E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 135–143, 1984.

[32] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, CA, 1979.

[33] M. D. Hansen. Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 604–610. IEEE, October 1989.

[34] D. S. Johnson, 1989. Personal communication.

[35] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983.

[36] D. T. Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE Transactions on Computers*, 31:478–487, 1982.

[37] T. Leighton and C. E. Leiserson. Wafer-scale integration of systolic arrays. *IEEE Transactions on Computers*, 34(5):448–461, May 1985.

[38] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 422–431. IEEE, October 1988.

[39] R. J. Lipton and R. E. Tarjan. A planar separator theorem. *SIAM Journal of Applied Mathematics*, 36(2):177–189, April 1979.

[40] C. H. Papadimitrio and M. Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM*, 29(2):285–309, April 1982.

[41] H. Rosenberger. Order-k voronoi diagrams for sites with additive weights in the plane. Technical Report UIUCDCS-R-88-1431, Department of Computer Science, University of Illinois, Urbana, IL, 1988.

[42] J. Spencer. *Ten Lectures on the Probabilistic Method.* SIAM, Philadelphia, PA, 1987.

[43] P. Vaidya. Geometry helps in matching. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 422–425, 1988.